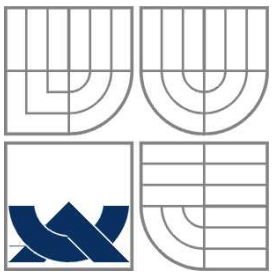


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ CESTY ROBOTU POMOCÍ POSILOVANÉHO UČENÍ

ROBOT PATH PLANNING BY MEANS OF REINFORCEMENT LEARNING

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **BC. MICHAL VESELOVSKÝ**
AUTHOR

VEDOUCÍ PRÁCE **RNDR. JIŘÍ DVOŘÁK, CSC.**
SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2012/2013

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Michal Veselovský

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Plánování cesty robotu pomocí posilovaného učení

v anglickém jazyce:

Robot path planning by means of reinforcement learning

Stručná charakteristika problematiky úkolu:

Úkolem plánování cesty robotu je nalezení cesty z počáteční do cílové pozice bez kolize se známými překážkami tak, aby ohodnocení cesty bylo minimální. Ohodnocení cesty je určeno hlavně délkou cesty a může zahrnovat i další aspekty, jako např. obtížnost a riziko cesty. Jedním z možných přístupů k řešení tohoto problému je posilované učení.

Cíle diplomové práce:

1. Analyzovat dosavadní přístupy k plánování cesty robotu.
2. Popsat problematiku posilovaného učení a jeho aplikace na plánování cesty robotu.
3. Navrhnout a implementovat systém pro plánování cesty robotu pomocí posilovaného učení.
4. Provést ověřovací a srovnávací experimenty.

Seznam odborné literatury:

BARRIOS-ARANIBAR, D; ALSINA, P. J. Reinforcement Learning-Based Path Planning for Autonomous Robots. In I ENRI - Encontro Nacional de Robótica Inteligente no XXIV Congresso da Sociedade Brasileira de Computação, Salvador, BA, Brazil, 2004.

JARADAT,M. A. K.; AL-ROUSAN, M.; QUADAN, L. Reinforcement based mobile robot navigation in dynamic environment. Robotics and Computer-Integrated Manufacturing, vol. 27,2011, pp. 135–149.

YU, J.-L. An adaptive gain parameters algorithm for path planning based on reinforcement learning. In Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou, 2005, pp. 3557-3562.

Vedoucí diplomové práce: RNDr. Jiří Dvořák, CSc.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2012/2013.

V Brně, dne 20.11.2012

L.S.

Ing. Jan Roupec, Ph.D.	prof. RNDr. Miroslav Doupovec, CSc., dr. h. c.
Ředitel ústavu	Děkan fakulty

ABSTRAKT

Tato práce se zabývá plánováním cesty pro autonomního robota v prostředí se statickými překážkami. Součástí práce je analýza různých přístupů k plánování cesty robota, a v implementační části popis metod využívajících posilovaného učení a experimenty s nimi. Hlavními výstupy práce jsou funkční algoritmy pro plánování cesty založené na Q-učení, ověření jejich funkčnosti a vzájemné srovnání.

ABSTRACT

This thesis is dealing with path planning for autonomous robot in enviromenment with static obstacles. Thesis includes analysis of different approaches for path planning, description of methods utilizing reinforcement learning and experiments with them. Main outputs of thesis are working algorithms for path planning based on Q-learning, verifying their functionality and mutual comparison.

KLÍČOVÁ SLOVA

Plánování cesty, posilované učení, Q-učení, robot, umělá inteligence

KEYWORDS

Path planning, reinforcement learning, Q-learning, robot, artificial intelligence

PROHLÁŠENÍ O ORIGINALITĚ

Prohlašuji, že jsem diplomovou práci na téma p**lánování cesty robotu pomocí posilovaného učení** vypracoval samostatně s použitím odborné literatury a pramenů, uvedených na seznamu, který tvoří přílohu této práce.

23.5.2013	Michal Veselovský
---	---
Datum	Jméno a příjmení

BIBLIOGRAFICKÁ CITACE

VESELOVSKÝ, M. *Plánování cesty robotu pomocí posilovaného učení.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 72 s. Vedoucí diplomové práce RNDr. Jiří Dvořák, Csc.

PODĚKOVÁNÍ

Rád bych poděkoval RNDr. Jiřímu Dvořákovi, CSc. za cenné připomínky a rady při vypracování diplomové práce a svým blízkým a rodině, kteří mi byli oporou.

Obsah:

1 ÚVOD

Během posledních let je vývoj chytrých a inteligentních zařízení stále populárnější a do jejich výzkumu jde více a více lidských i finančních zdrojů. K těmto zařízením se řadí i stále sofistikovanější roboty, ať už jde o stacionární roboty (manipulátory) využitelné v průmyslové výrobě nebo o mobilní autonomní roboty, které mají velice variabilní využití.

Tyto mobilní autonomní roboty jsou schopné pracovat za různých okolních podmínek, vykonávat činnosti pro člověka nezaživné a repetetivní, a dokáží takto pracovat po dlouhou dobu, spolehlivě a bez zásahu člověka. Jejich využití může být různé, například hlídací a sledovací zařízení, průzkumní roboti, nebo například mobilní reklamní bannery.

Jedním z nejdůležitějších problémů u autonomních robotů je řízení jejich pohybu. Řízení pohybu znamená rozhodování jakou posloubnost pohybů zvolit tak, aby robot dosáhl svého cíle. Řízení pohybu sestává z navigování robota v konečném čase z počáteční do koncové pozice a bez kolize s jakýmikoliv překážkami. Klasický způsob řešení tohoto problému je rozdělení na menší podproblémy, konkrétně plánování cesty, generování trajektorie a provedení pohybu po trajektorii.

Pro řešení problému plánování cesty existuje více různých metod, mezi nejběžnější patří například metoda potenciálových polí, metody založené na rozkladu do buněk nebo na mapách cest. Tyto metody jsou stručně popsány v první části práce.

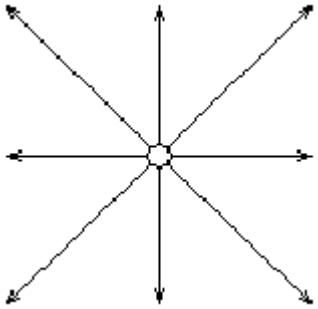
Nicméně tyto metody mají i své nevýhody a omezení, a proto se při řešení problémů pohybu a plánování stále častěji využívají metody umělé inteligence. Jedno z možných řešení je pomocí metod posilovaného učení, konkrétněji například tzv. Q-učení (Q-learning).

2 SEZNÁMENÍ S PROBLEMATIKOU

2.1 Reprezentace prostředí

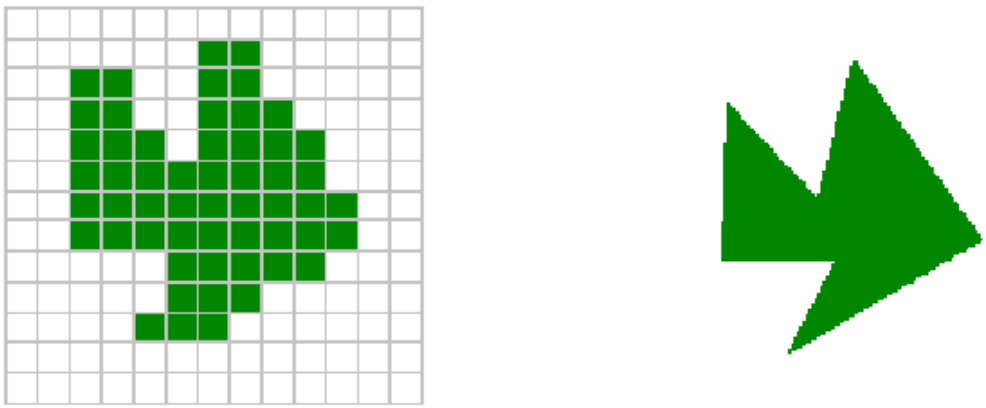
2.1.1 Typy prostředí

- Diskrétní prostředí – jedním z možných způsobů reprezentace diskrétního prostředí je rozdělení do buněk obvyklé stejných nebo i různých velikostí. Počet a velikost buněk je možno zvolit dle požadované přesnosti plánování, dosahu senzorů, velikosti robota (buňka nesmí být menší než je velikost robota), prostředí a překážek. Platí, že čím více buněk, tím vyšší je přesnost a členitost, ale také se zvyšuje výpočetní i paměťová náročnost. Směr pohybu robota je často omezen typem diskretizace (při diskretizaci do čtvercových buněk má robot obvykle k dispozici pouze 4 až 8 směrů pohybu). Více informací o diskrétních prostředích viz kapitola 3.3.2.



Obr. 1 Nejčastější možné směry pohybu v diskrétním prostředí.

- Spojité prostředí – umožňuje přesnější reprezentaci prostředí a překážek, jelikož není rozděleno do buněk, ale je souvislé. Je tedy možné volit naprosto libovolné směry pohybu, reprezentace prostředí je celkově přesnější a více podobná reálnému světu, a výsledná cesta může být plynulejší a často lepší. Nevýhodou je však obvykle mnohem vyšší výpočetní náročnost algoritmů pro tento typ prostředí [4].



Obr. 2 Reprezentace překážek v diskrétním a spojitém prostředí [20]

Spojitá prostředí lze diskretizovat pomocí různých metod, např. pomocí voronoiových diagramů, rozdělením do čtvercové mřížky, pravděpodobnostními metodami atd. Takto zdiskretizované prostředí pak lze popsat následujícími způsoby:

2.1.2 Stavový prostor

Stavový prostor X je základním prostředkem popisu diskrétního prostředí.

$$X = [S, A, s_0, C] \tag{1}$$

kde:

- S je množina všech stavů
- A je množina všech možných akcí
- s_0 je počáteční stav
- C je množina koncových stavů

Nejběžnějším způsobem reprezentace stavového prostoru je orientovaný graf, ve kterém uzly reprezentují jednotlivé stavy, a orientované hrany mezi nimi reprezentují možné akce. Problém plánování se takto transformuje na problém hledání cesty v grafu [4].

2.1.3 Pracovní prostor

Zjednodušený model stavového prostoru se nazývá pracovní prostor. Pracovní prostor W je prostor, ve kterém se robot může pohybovat a provádět akce.

Obvyklým způsobem reprezentace je N rozměrný (N = 2,3) euklidovský prostor R^N . V tomto prostoru se mohou vyskytovat statické i dynamické překážky omezující možnosti pohybu robota [4].

2.1.4 Konfigurační prostor

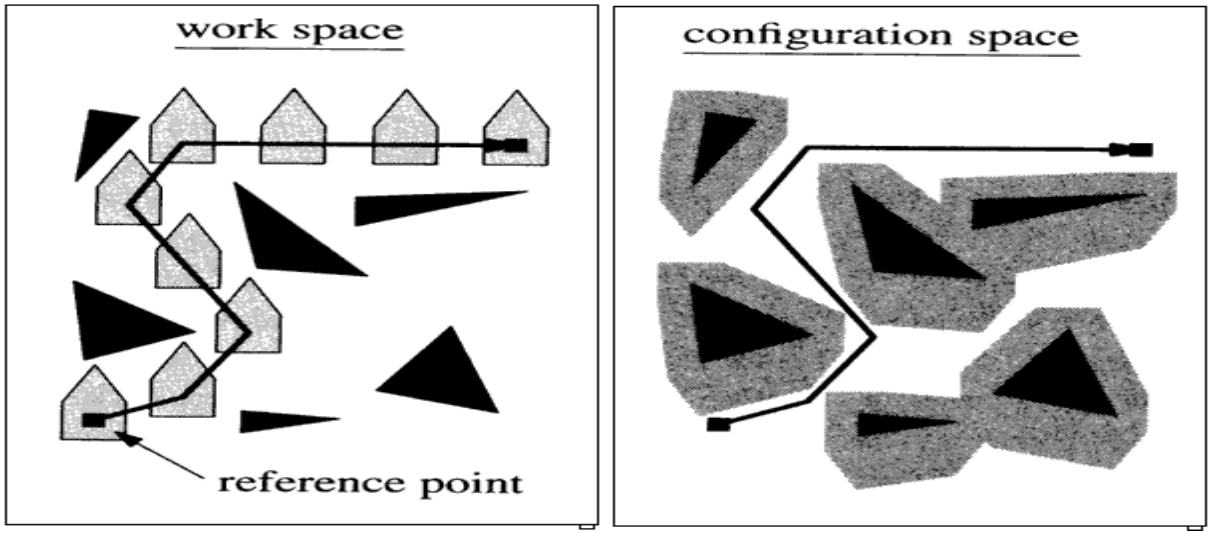
Pro další zjednodušení stavového prostoru je možné zavést pojem konfigurační prostor C. Tento prostor usnadňuje matematický popis polohy a orientace robota v pracovním prostoru. Konfigurační prostor je n-dimenzionální prostor, kde n je počet parametrů jednoznačně definujících pozici/konfiguraci robota.

Konfigurace robota q je obvykle popsána vektorem, jehož složky určují pozici a orientaci robota v konfiguračním prostoru.

Volný konfigurační prostor C_{free} je prostor všech konfigurací, které nekolidují s žádnou překážkou a vyhovují všem omezením kladeným na robota – tedy všechny přípustné pozice robota. Je nezbytné, aby startovní a cílová pozice robota ležely ve volném konfiguračním prostoru [11].

Kolizní konfigurační prostor je naopak ta část stavového prostoru, jež je obsazena překázkami a jde tedy o prostor C_{obst}.

Jednou z hlavních myšlenek konfiguračního prostoru je možnost reprezentace robota jako bodu namísto tvarově složitého tělesa. Tato redukce robota na bod spočívá ve virtuálním zvětšení překážek o rozměry robota. Problém plánování cesty robota je tedy převeden na problém plánování pohybu bodu ve volném konfiguračním prostoru C_{free} [4].



Obr. 3 Převod pracovního prostoru W do konfiguračního prostoru C_{free} a redukce robota na bod [21]

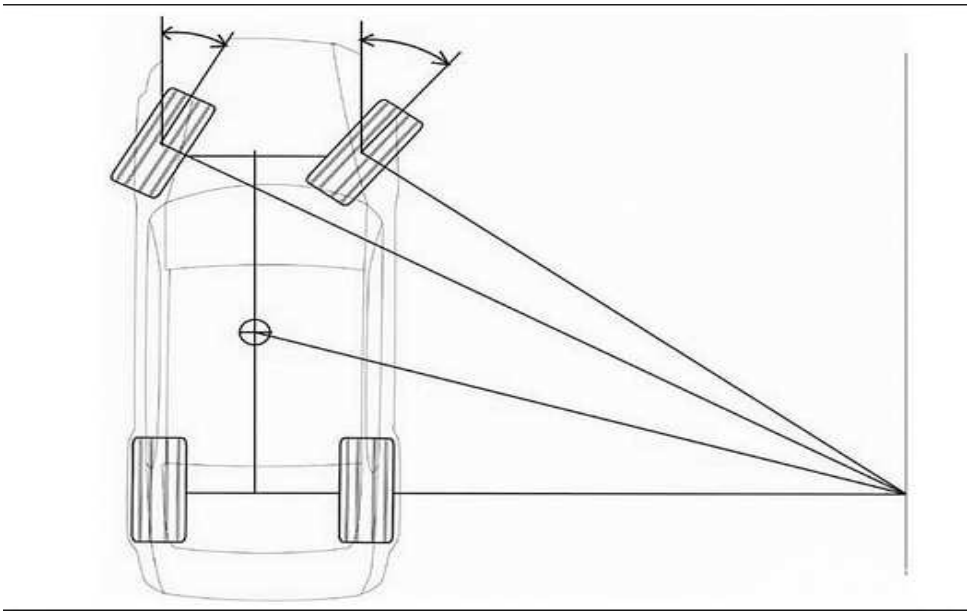
2.2 Omezení pohybu kladené na roboty

Problém plánování cesty lze podle omezení pohybu kladených na roboty rozdělit na dva druhy – holonomní a neholonomní plánování.

V případě robota se jako holonomní označuje takový robot, jehož počet řiditelných stupňů volnosti je stejný jako celkový počet stupňů volnosti. Neholonomní robot je pak takový, jehož počet řiditelných stupňů volnosti je menší než celkový počet stupňů volnosti.

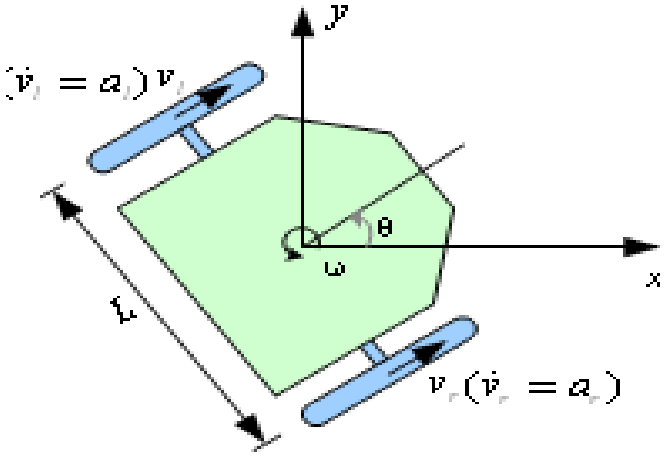
Neholonomní pohyb reprezentuje například:

- Automobil - robot s takzvaným Ackermanovým řízením. V tomto případě robot nemůže měnit svou orientaci přímo, ale je nutné změnit natočení kol, a následně vykonat pohyb vpřed či vzad.



Obr. 4 Ackermanovo řízení u automobilu

- Tank – robot s diferenciálním řízením. Do této kategorie spadají roboty schopné otáčet se na místě kolem své osy, nejsou ale schopny přímého pohybu do strany – je nutné nejprve změnit orientaci a následně vykonat pohyb vpřed nebo vzad. Změna orientace je určena rozdílem rychlostí levého a pravého kola/pásu.



Obr. 5 Diferenciální řízení [22]

Holonomní roboty jsou pak obecně všesměrové roboty, které se mohou pohybovat libovolným směrem nezávisle na svém aktuálním natočení a otáčet se kolem své osy bez omezení. Tomuto způsobu pohybu se také říká „omnidrive“ [4].

Obr. 6 Základní model podvozku všesměrového robota [23]

3 ANALÝZA DOSAVADNÍCH PŘÍSTUPŮ PLÁNOVÁNI CESTY

3.1 Dělení metod

Při plánováni pohybu je úkolem agenta vytvořit si model prostředí, a s pomocí tohoto modelu vytvořit plán svých akcí tak, aby se dostal ze svého počátečního stavu do zadaného cílového stavu a to co nejlepší cestou a bez kolize s překážkami.

Pro plánování cesty robotů existuje několik základních metod a mnoho různých modifikací či kombinací. Tyto metody lze obvykle rozdělit dle několika kritérií. K nejdůležitějším patří dělení dle typu prostředí a dělení na globální a lokální metody.

Dělení metod dle prostředí a typu překážek:

a) Metody pro plánování se statickými překážkami ve známém prostředí.
b) Metody pro plánování se statickými překážkami v neznámém nebo v částečně známém prostředí.
c) Metody pro plánování s dynamickými překážkami ve známém prostředí.
d) Metody pro plánování s dynamickými překážkami v neznámém nebo v částečně známém prostředí.

Globální a lokální plánování:

a) Globální plánování – toto plánování se provádí ještě před samotným vykonáním pohybu robota. Jeho úkolem je nalézt nekolizní cestu z počáteční pozice do cílové, proto před samotným plánováním vyžaduje model daného pracovního prostředí. Problém tohoto plánování je vysoká výpočetní náročnost, především u robotů s více stupni volnosti a u rozsáhlých prostředí.
b) Lokální plánování – toto plánování má na starosti vlastní řízení pohybu robota po dané cestě. Tato cesta může vést přímo k cíli, a plánovač se pak dynamicky vyhýbá nalezeným překážkám, a nebo může vést po trase naplánované globálním plánováním. Lokální plánovač také musí zohledňovat případná omezení (omezení pohybu robota, nové překážky apod.). Lokální plánovače jsou obecně výrazně rychlejší, ale občas nemusí nalézt optimální cestu, přestože taková existuje. Častý problém bývá uváznutí v lokálních optimech, ze kterých je obtížné pokračovat k cíli.

Plánováni cesty obvykle sestává ze dvou hlavních kroků:

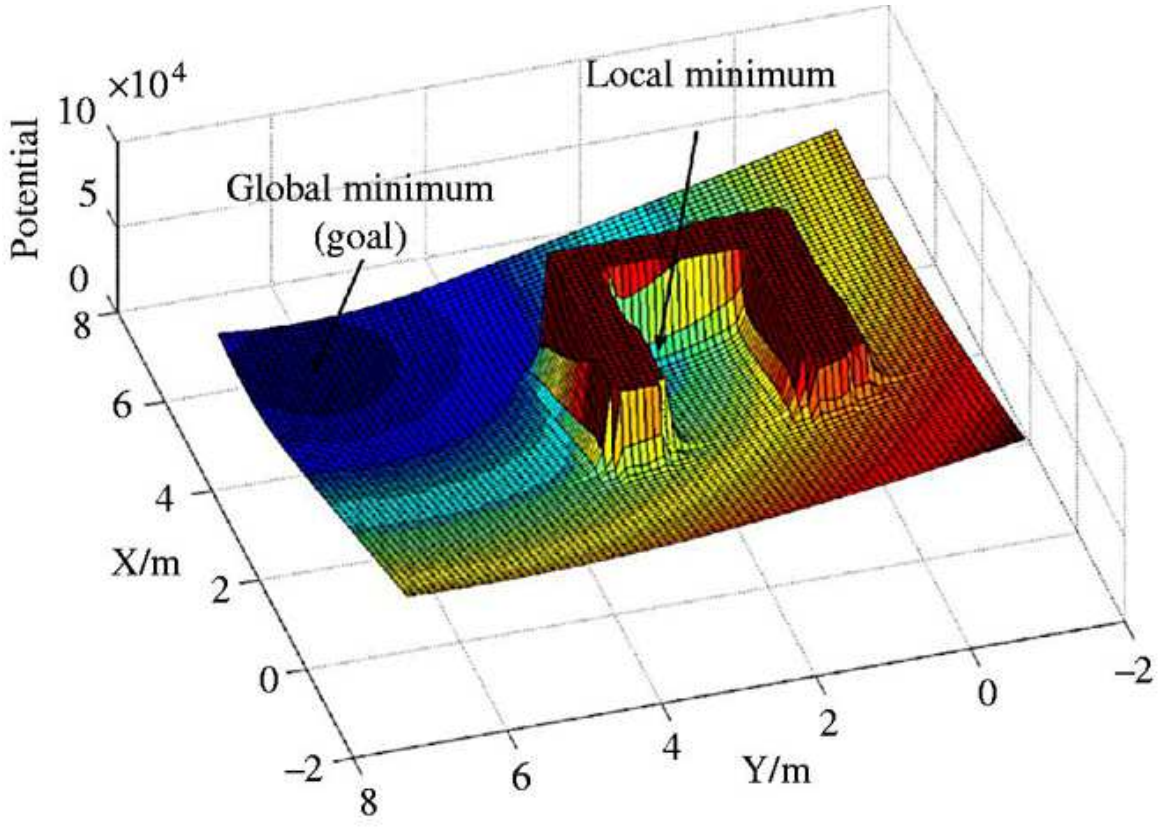
- **předzpracování:** reprezentace volného konfiguračního prostoru C_{free} mezi veškerými překážkami v oblasti pomocí grafů či funkcí
- **zpracování dotazu:** prohledávání grafu nebo použití funkce pro nalezení cesty z počátečního bodu do koncového [3]

3.2 Metoda potenciálových polí (potential field method)

Předzpracováni spočívá v umístění mřížky nad oblast a definici funkce potenciálového pole $E(x,y)$. Cílový bod má nejnižší potenciál, počáteční bod má potenciál vyšší, místa kde jsou umístěny překážky mají potenciál vyšší než jejich okolí, čímž tvoří neprůchodné „kopce“ (viz obr. 7).

Zpracování dotazu zahrnuje hledání cesty do globálního minima potenciálového pole. Agent je tedy v podstatě „přitahován“ ke svému cíli a „odpuzován“ od překážek. Mezi

výhody patří rychlost nalezení cílové konfigurace, zásadní nevýhodou je však možnost uváznutí v lokálním minimu [3].



Obr. 7 Metoda potenciálových polí [24]

3.3 Metody rozkladu do buněk (cell-decomposition)

Rozklad do buněk zahrnuje několik různých metod. Při předzpracování je prostor rozložen do buněk různého tvaru v závislosti na metodě. U buněk se určí, jestli obsahují nebo neobsahují překážku, a vytvoří se graf sousednosti. Vrcholy grafu jsou buňky neobsahující překážky a hrany jsou tvořeny spojnicemi mezi těmito vrcholy.

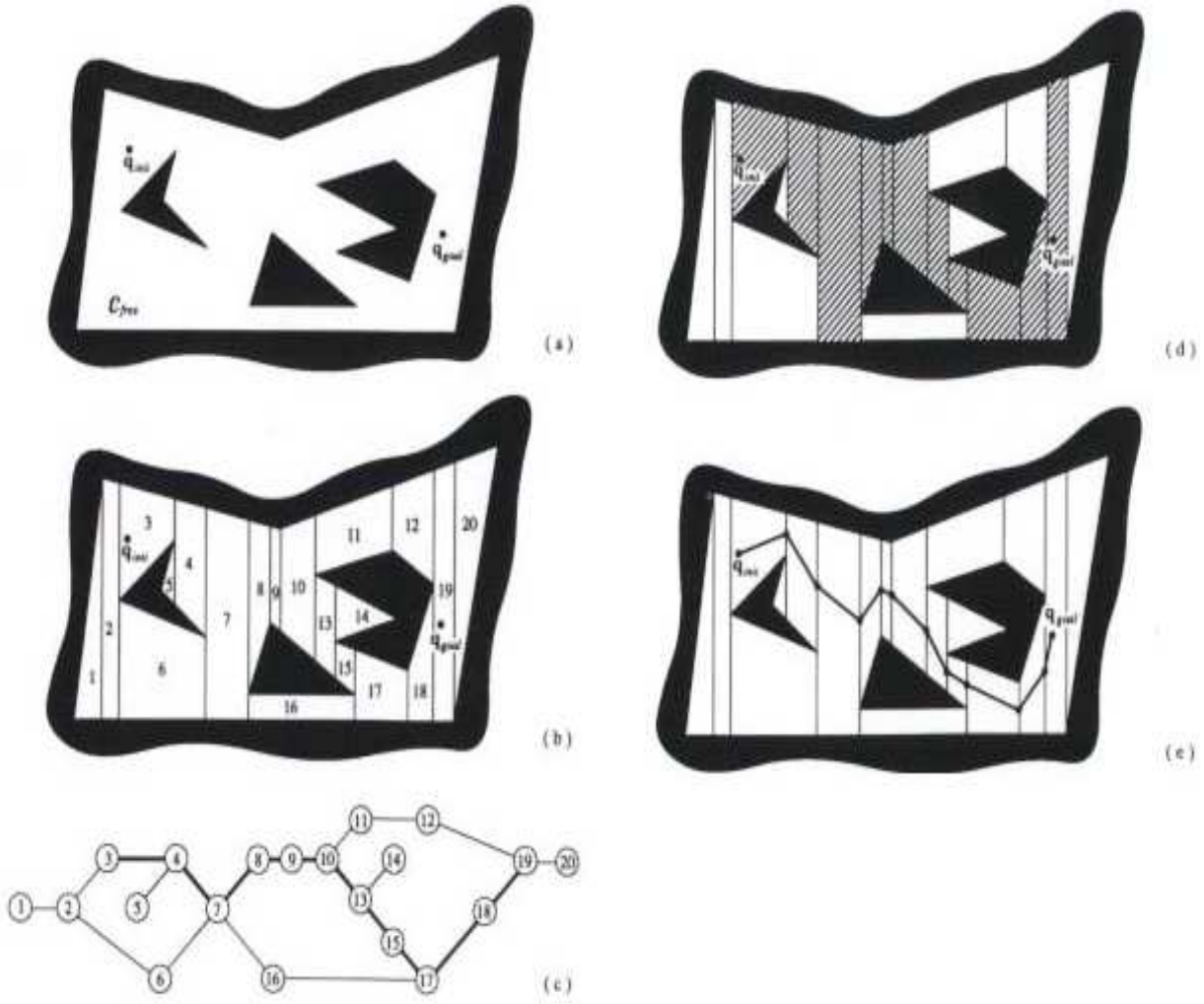
Zpracování dotazu sestává z nalezení posloupnosti sousedních buněk vedoucí z počáteční konfigurace do cílové konfigurace, a transformace této posloupnosti do cesty [3].

Běžně se používají dvě základní metody rozkladu do buněk; exaktní rozklad a aproximativní rozklad.

3.3.1 Exaktní rozklad

U této metody se konfigurační prostor C_{free} rozloží do vzájemně se nepřekrývajích buněk jednoduchého tvaru (obvykle lichoběžníky nebo trojúhelníky). Určí se body přechodu, jež jsou vždy na hranici mezi buňkami.

Cesta se pak skládá z počátečního a cílového bodu a z bodů přechodu. Metoda patří mezi úplné metody, tzn. vždy nalezne cestu (pokud existuje) a nebo potvrdí její neexistenci [4].

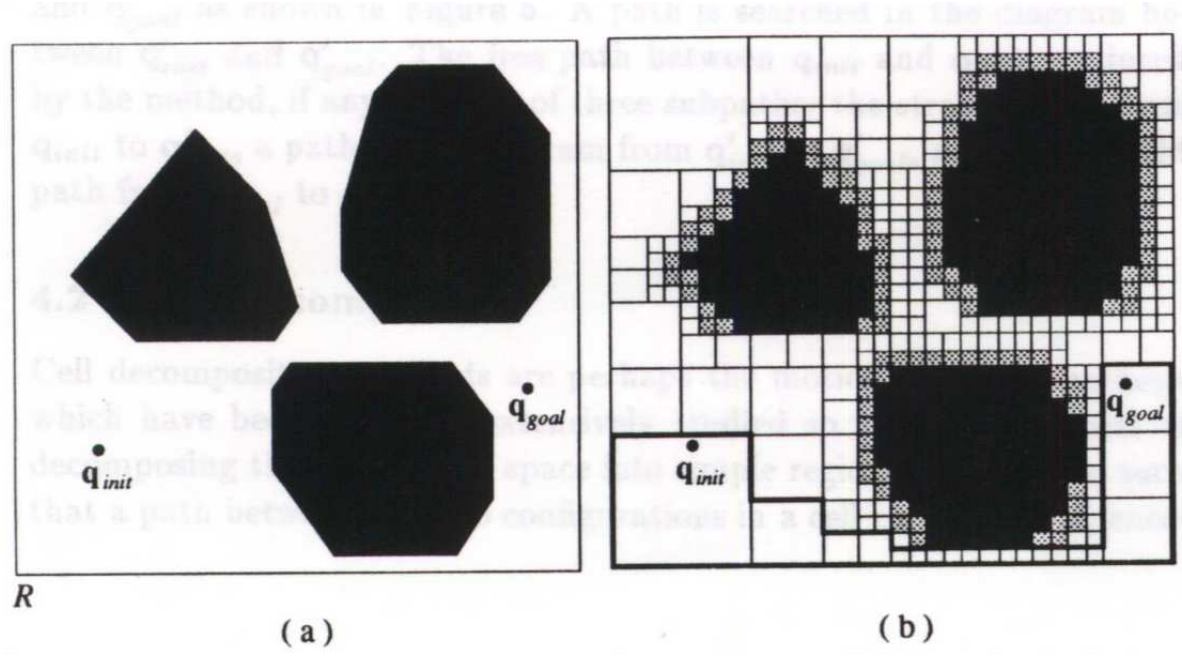


Obr. 8 Exaktní metoda rozkladu do buněk [25]

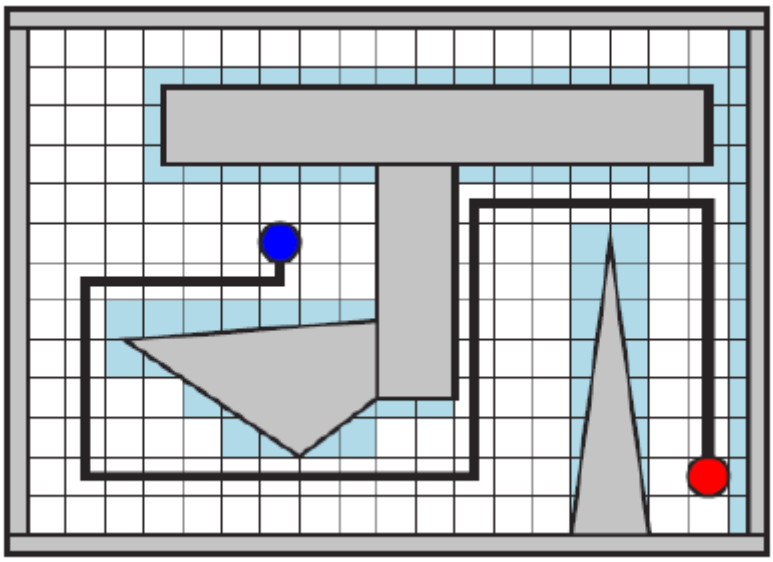
3.3.2 Aproximativní rozklad

Celý prostor včetně překážek je rozdělen na pravidelné (obvykle čtvercové) buňky, čímž převedeme spojitý prostor na diskrétní. Při rozkladu je důležité vhodně určit velikost buněk – menší buňky zpřesňují rozklad, nicméně také zvyšují výpočetní i paměťovou náročnost. Buňky obvykle mají stejnou velikost, není to však podmínkou.

U buněk se rozlišuje, zda-li je obsazená (pokud do buňky zasahuje alespoň jeden bod překážky) nebo neobsazená (neobsahuje žádné body překážky). Tento způsob patří mezi metody neúplné; nemusí najít žádnou cestu i v případě že cesta existuje [4]. Tento problém se dá minimalizovat co nejvyšší přesností rozkladu (zmenšením buněk).



Obr. 9 Aproximativní rozklad do různě velkých buněk [25]



Obr. 10 Aproximativní rozklad do stejně velkých buněk a nalezení cesty [5]

3.4 Metoda mapy cest (road-map)

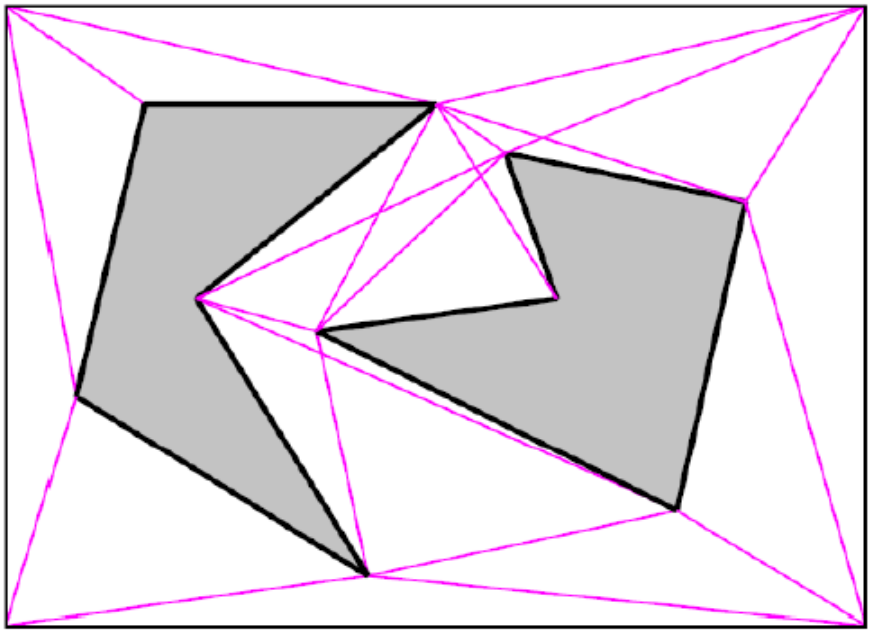
Krok předzpracování spočívá v konstrukci grafu cest (road-mapy), který reprezentuje volný konfigurační prostor C_{free}. Vrcholy jsou tvořeny body v pracovním prostoru v závislosti na zvolené metodě. Spojnice mezi těmito body, reprezentující cesty, po kterých se robot může pohybovat, tvoří hrany grafu.

Zpracování dotazu zahrnuje přídání počátečního a koncového bodu do grafu, což převede úlohu na problém hledání nejkratší cesty v grafu pomocí známých algoritmů, např. Dijkstrův algoritmus nebo A* [3]. Velkou výhodou je úplnost algoritmu. Pokud tedy cesta existuje, algoritmus ji vždy najde.

3.4.1 Deterministické metody

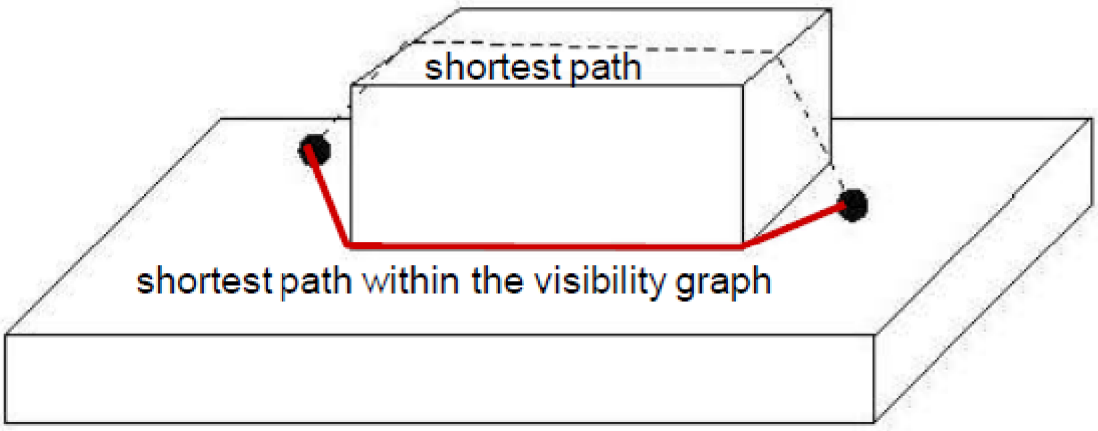
Graf viditelnosti

Vrcholy grafu jsou tvořeny počátečním bodem, koncovým bodem a vrcholy všech překážek v pracovním prostoru. Spojnice těchto vrcholů, které neprochází žádnou překážkou pak tvoří hrany grafu. Hrany mohou vést i podél překážek [6]. Pokud jsou překážky tvořeny křivkou (kruh, elipsa, zakřivené hrany), je nutné je převést na polygony.



Obr. 11 Graf viditelnosti [5]

Tento přístup má několik nevýhod; nedokáže najít optimální cestu ve 3D prostoru (viz obr. 12), navíc nalezené cesty nejsou vždy ideálně volné, protože mohou vést po obvodu překážek, což může způsobovat komplikace [6].

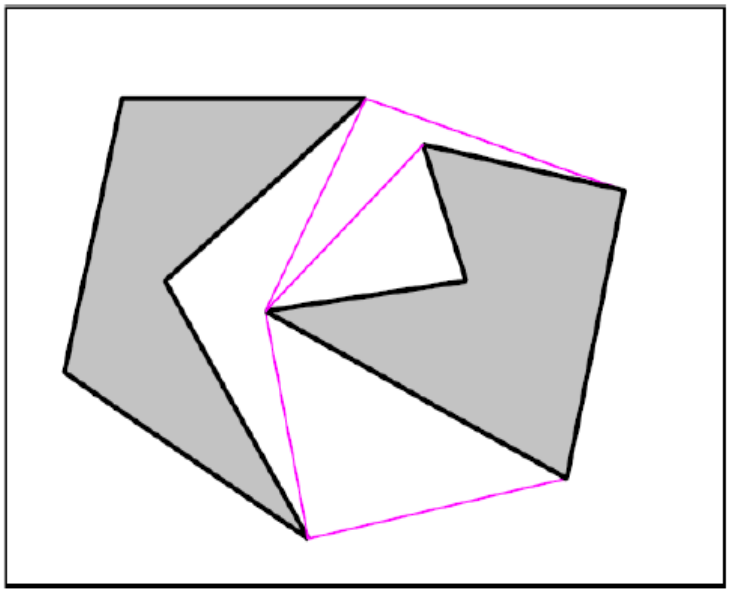


Obr. 12 Graf viditelnosti ve 3D prostoru [6]

Graf tečen

Jedná se o redukovaný graf viditelnosti, ve kterém jsou ponechány pouze hrany tvořené tečnami jednotlivých vrcholů. Oproti grafu viditelnosti umožňuje rychlejší výpočet

díky menšímu počtu hran, a je možné jej použít i na překážky tvořené křivkami bez převodu na polygony.

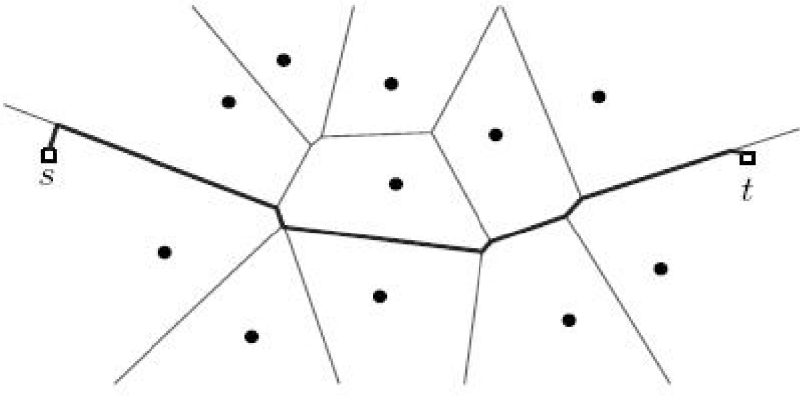


Obr. 13 Graf tečen [5]

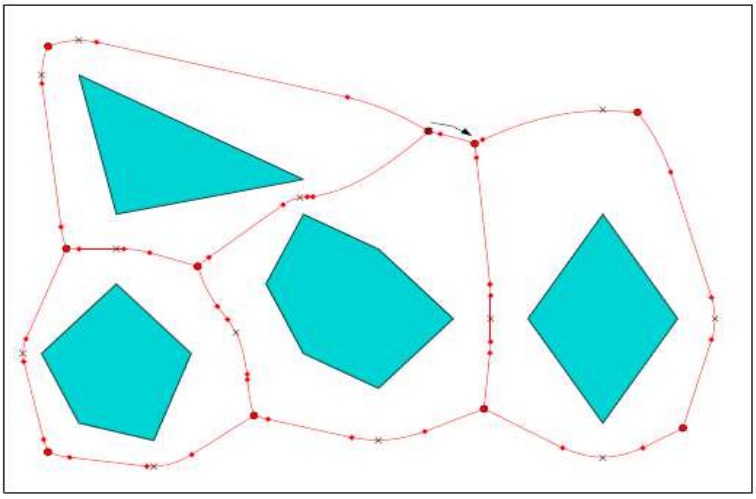
Voronoiovy diagramy

Diagram je tvořen body v rovině, jež mají stejnou vzdálenost od dvou nebo více nejbližších překážek. Množiny těchto bodů tvoří hrany diagramu. Tyto hrany reprezentují cesty po kterých se může robot bezpečně pohybovat bez rizika kolize s překážkami. Při převádění diagramu na graf jsou vrcholy tvořeny body, které mají stejnou vzdálenost od tří a více překážek, a dále počátečním a koncovým bodem. Hrany jsou spojnice mezi těmito body.

Řešení problému spočívá v nalezení nejkratší cesty grafem pomocí některého ze známých algoritmů. Tato metoda generuje velice bezpečné cesty, které zaručují maximální možnou vzdálenost od nejbližší překážky [6].



Obr. 14 Voronoiuv diagram s bodovými překážkami [26]



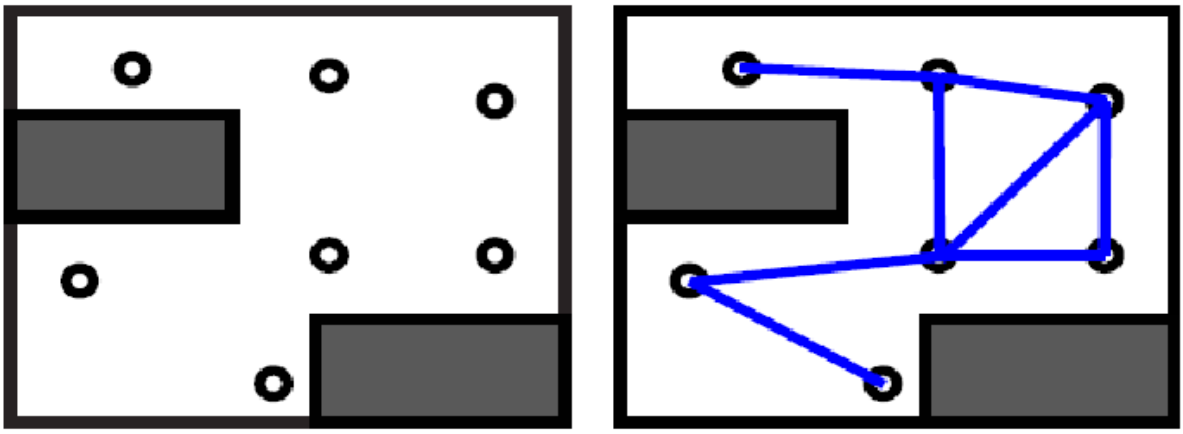
Obr. 15 Voronoiův diagram s polygonovými překážkami [5]

3.4.2 Pravděpodobnostni metody

Pravděpodobnostní mapy cest (Probabilistic Road Maps - PRM)

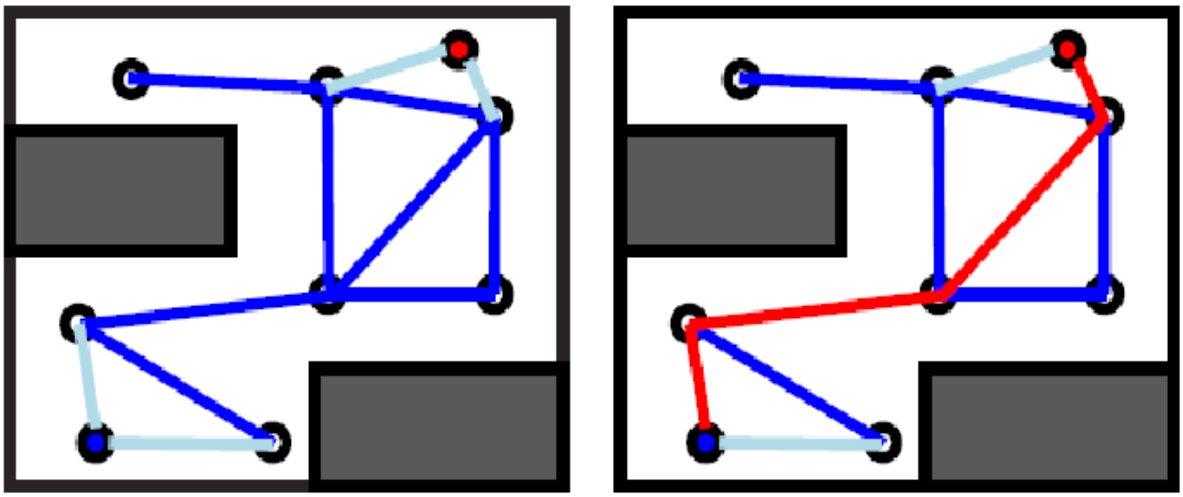
Algoritmus sestává ze dvou fází. V první, učící fází, se generuje mapa cest reprezentovaná pomocí grafu, ve druhé fázi se hledá nejkratší cesta grafem

V učící fázi se vytváří mapa cest popisující spojitost konfiguračního prostoru [6]. Náhodně se vybere místo v pracovním prostoru. Pokud je místo ve volném konfiguračním prostoru C_{free} (neleží v žádné překážce), tak je místo přidáno jako vrchol grafu. Algoritmus se následně snaží propojit nový vrchol s ostatními a pokud mezi nimi neleží žádná překážka, do grafu se přidá nová hrana. Tento proces se neustále opakuje dokud není dostatečně popsán celý konfigurační prostor nebo nedojde pamět či čas vyhrazený pro výpočet.



Obr. 16 Učící fáze pravděpodobnostní metody [5]

Fáze hledání cesty začíná přípojením počátečního a koncového bodu do grafu stejným způsobem jako v učící fázi. Následně se použije některý ze známých algoritmů hledání nejkratší cesty v grafu. Algoritmus nemusí vždy najít existující cestu, což je obvykle způsobeno nedostatečným pokrytím konfiguračního prostoru *C*. V těchto případech je možné opakovat první fázi znovu, do doby než bude pokrytí dostatečné.



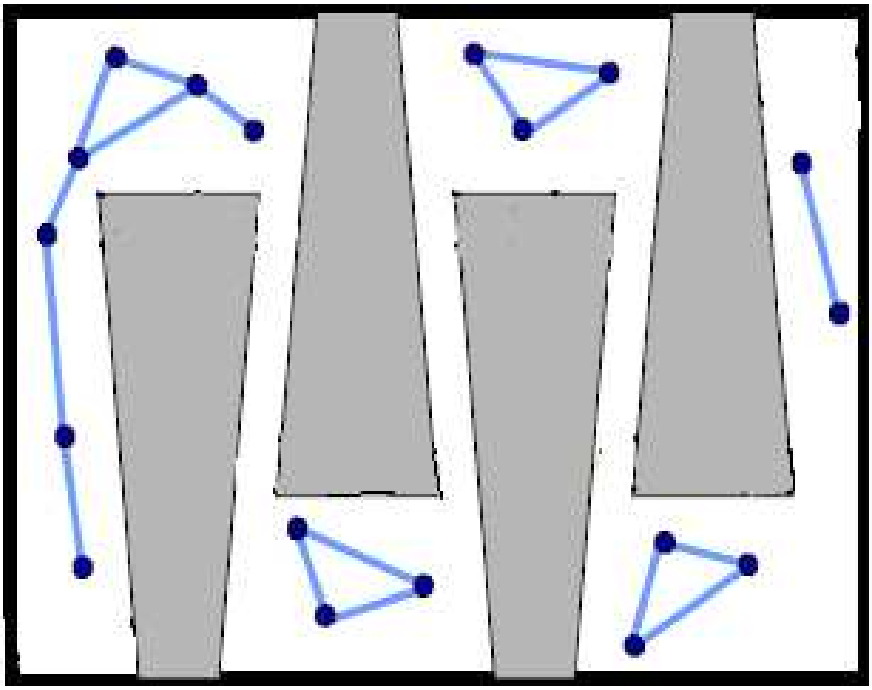
Obr. 17 Fáze hledání cesty podle pravděpodobnostní metody [5]

Výhody:

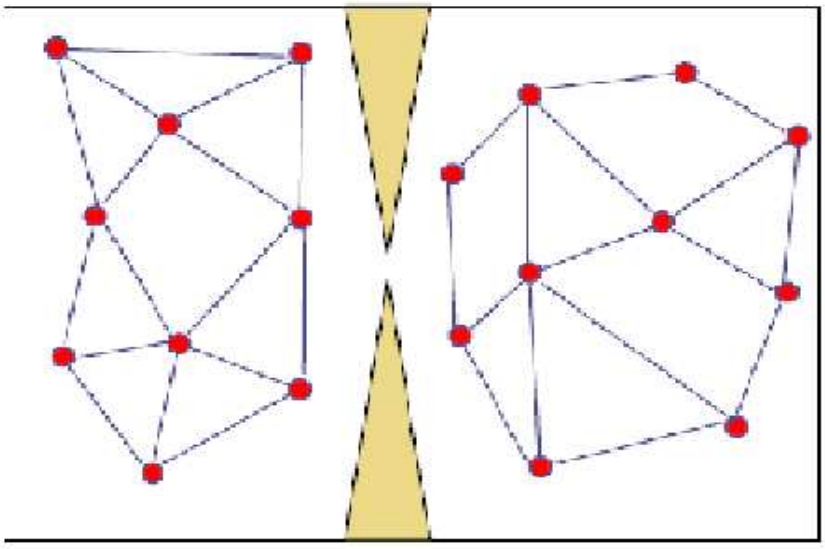
- s vhodným testováním dostatečného pokrytí je metoda pravděpodobnostně úplná
- snadno aplikovatelné na trojrozměrný pracovní prostor
- s dostatečným předzpracováním dokáže hledat cesty velice rychle [6]

Nevýhody:

- problémy s nalezením cesty v případě úzkých průchodů kvůli malé pravděpodobnosti umístění vrcholu do tohoto průchodu; může tak vzniknout nesouvislý graf [6]
  Tento problém je nutné vyřešit dodatečnou metodou



Obr. 18 Možné problémy pravděpodobnostní metody [6]

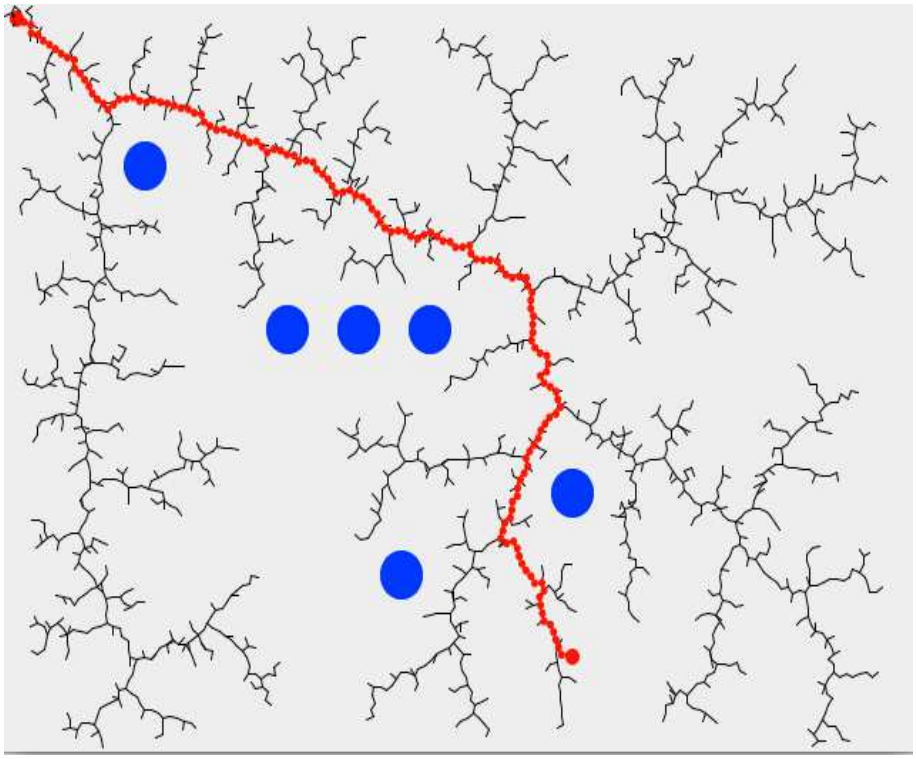


Obr. 19 Možné problémy pravděpodobnostní metody [4]

Pravděpodobností stromy (Rapidly-exploring Random Trees – RRT)

Algoritmus se snaží rychle a co nejrovnoměrněji prohledávat pracovní prostor robota. Inkrementálně vytváří prohledávací strom tak, že při každé iteraci se strom rozroste o další vrchol náhodným směrem. Jednotlivé vrcholy pak představují možné pozice robota a hrany jeho možné akce.

Metoda je velice vhodná především pro plánování cesty neholonomních robotů, nicméně má i své nevýhody. Hlavním problémem je především příliš pomalá konvergence k cíli, protože se stromy rozrůstají rovnoměrně na všechny strany. Běžně se tedy používají modifikace metody, uzpůsobené konkrétní aplikaci [5].



Obr. 20 Hledání cesty pomocí pravděpodobnostních stromů (RRT) [27]

4 POSILOVANÉ UČENÍ

Posilované učení (Reinforcement learning - RL) je inspirováno chováním živých tvorů. Jako příklad může posloužit myš zavřená v kleci s dvěmi tlačítky a s elektrodami na zádech. Myš je tvor zvídavý, a tak se po chvíli bloudění po kleci zajisté dostane ke tlačítkům. Pokud myš zmáčkne první z tlačítek, dávkovač jí do klece hodí pamlsek jako odměnu. Pokud však zmáčkne druhé z tlačítek, dostane za trest malý elektrický šok. Myš se tak dokáže velice rychle naučit, které z tlačítek má mačkat aby maximalizovala svou odměnu, a nedostala žádný trest.

U posilovaného učení se tedy agent učí, jaké akce provést, aby maximalizoval celkovou odměnu nebo minimalizoval celkový trest za své akce v určitém časovém horizontu. Odměna/trest jsou obvykle vyjádřeny jako číselná hodnota přiřazená ke každé akci nebo výslednému stavu. Agent neví, jaká akce poskytuje nejvyšší možnou odměnu, tudíž se to vyzkoušením všech možných akcí snaží zjistit.

Posilované učení patří mezi metody strojového učení bez učitele. To znamená, že agent se učí získáváním odměny/trestu přímo od svého prostředí, bez jakéhokoliv dohledu, který by ho informoval, jestli provádí správnou nebo špatnou akci. Jediný takový indikátor je jeho vlastní strategie vytvářená během samotného učení. Z toho vyplývá, že agent je při vložení do cizího prostředí schopný se učit naprosto samostatně, bez dalších tréninkových dat nebo dodatečných údajů. Na druhou stranu je tento způsob učení často výpočetně i časově náročnější.

4.1 Složky posilovaného učení

Mezi základní složky posilovaného učení patří:

- množina stavů S
- množina akcí A
- transformační funkce δ: S x $A \rightarrow S$
- odměna r: S x $A \rightarrow R$
- strategie chování π: $S \rightarrow A$
- hodnotová funkce V^{π}: $S \rightarrow R$

Hodnotová funkce V^{π} je určena následnovně:

$$V^{\pi}(s_t) \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{2}$$

γ: $0 \leq \gamma < 1$ – srážkový faktor. Hodnota blízká 0 znamená, že agent více dbá na okamžité odměny. Při hodnotě bližící se 1 si agent více cení budoucích odměn. Hodnota srážkového faktoru musí být menší než 1 aby mohl algoritmus konvergovat.

Optimální strategii chování π^* je vyjádřena takto :

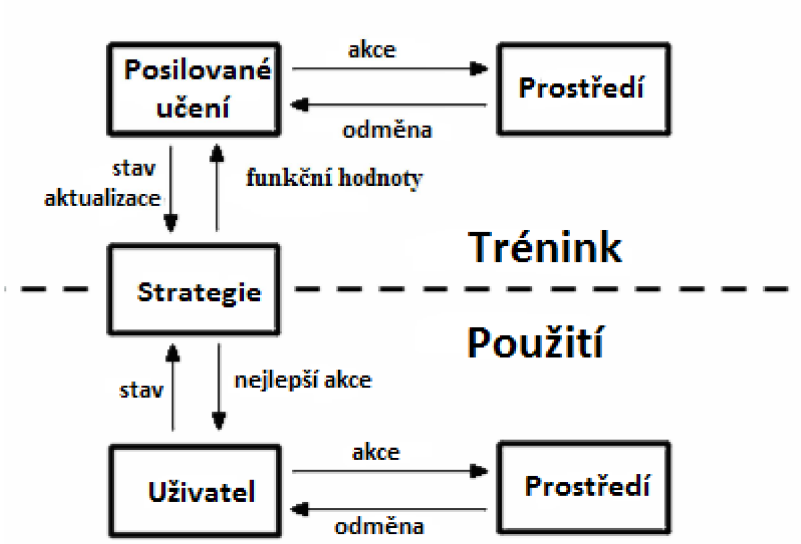
$$\pi^* \equiv \operatorname{argmax}_{\pi} V^{\pi}(s) \tag{3}$$

a optimální akce pro stav s je [10]:

$$\pi^*(s) \equiv \operatorname{argmax}_{\pi}(r(s,a) + \gamma V^*(\delta(s,a))) \tag{4}$$

4.2 Řešení problémů pomocí posilovaného učení

Řesení problému pomocí metod posilovaného učení má obvykle dvě části. První část je trénovací, kdy je pomocí algoritmů posilovaného učení vytvářena rozhodovací strategie π(s) pro dané prostředí. Ve druhé části je získaná strategie π(s) aplikována pro řešení daného problému výběřem vhodné akce v závislosti na strategii a současném stavu; $a = \pi(s)$. Interakce obou fází je naznačena na obr. 21.



Obr. 21 Využití strategie (policy) získané posilovaným učením

V některých případech se tyto dvě fáze mohou navzájem prolínat. Taková situace je pak nazývána „učením za chodu“. Např. několika málo opakováními je naučena pouze základní strategie, která je dále postupně vylepšována při reálném provozu. Takové řešení samozřejmě není vhodné za všech okolností z důvodu výrazně horších počátečních výsledků. Je tedy nutné zhodnotit zda je tento přístup bezpečný a žádoucí.

4.3 Základní principy posilovaného učení

Principem posilovaného učení je provádění akcí a sledování jejich následků. Tento princip je také znám jako „příčina a následek“ a je poměrně snadné převést jej do několika základních kroků.

1. Agent je postaven do výchozího stavu s_t.
2. Agent vybere následující akci a_t z množiny možných akcí A. Akci vybírá buďto náhodně, nebo dle předem definované strategie.
3. Agent provede akci a_t vedoucí do následujícího stavu s_{t+1}.
4. Agent od prostředí obdrží odměnu nebo trest (posílení) $r(s_t, a_t)$ za svou akci.
5. Informace o posílení je zaznamenána pro daný pár stav/akce (s_t, a_t) [7].

Ohodnocení $r(s_t, a_t)$ může být kladné, neutrální nebo záporné. Snahou agenta je maximalizovat součet těchto odměn $\sum \gamma^t r(s_t,a_t)$. Opakovaným prováděním akcí a pozorováním následných odměn je postupně získávána strategie určující nejlepší akci pro daný stav. Po dostatečném počtu opakování tohoto postupu agent nakonec dojde k optimální celkové strategii π^* pro dané prostředí [7].
Cílem posilovaného učení je tedy najít optimální strategii π^*, díky níž ve vykonávací fázi můžeme určit nejvhodnější akci $a = \pi(s)$. Pro nalezení optimální strategie je zavedena

hodnotová funkce $V^{\pi}(s)$ strategie π , která udává očekávanou odměnu při počátečním stavu s a použití strategie π [8].

4.4 Strategie pro výběr následující akce

Jednou z důležitých otázek při návrhu systému posilovaného učení je správná volba strategie pro výběr následující akce. Je totiž nutné provést kompromis mezi prohledáváním co největšího prostoru a využíváním dosud nejefektivnějších nalezených hodnot. Pokud agent dostal odměnu za nějakou akci v minulosti, má tendenci si příště zvolit stejnou akci, aby opět dostal odměnu. Agent tak využívá svých znalostí k dosažení co největší odměny, avšak zanedbává prozkoumávání ostatních akcí, při kterých však odměna může být vyšší. Aby se agent učil správným způsobem, je nutné najít rovnováhu mezi oběma přístupy [7].

Základní strategie pro výběr akce:

- ε–hladová (greedy) strategie – Většinou je vybírána akce s nejvyšším možným ziskem, ale občas, s malou pravděpodobností ε je vybrána naprosto náhodná akce. Tato metoda zajišťuje, že s nekonečným množstvím pokusů bude každá akce vyzkoušena nekonečněkrát, čímž je zajištěno nalezení optimální funkce [7].
- Softmax strategie – přiřazuje váhy jednotlivým akcím na základě jejich odměn. Díky tomu akce s nejvyšší odměnou mají největší šanci že budou vybrány, zatímco akce s nízkou hodnotou nebo dokonce trestem mají šanci velice nízkou. Vhodné především pro situace kdy je nejhorší možná varianta velice nežádoucí.
- Náhodný výběr (prohledávací strategie)

4.5 Q-učení

Jedním z nejdůležitějších průlomů v posilovaném učení, byl vývoj algoritmu známého jako Q-učení (Watkins, 1989). Pro deterministické prostředí je určen vztahem:

$$Q(s_t,a_t) \leftarrow r_{t+1} + \gamma \max_{\alpha} Q(s_{t+1}, a) \tag{5}$$

a pro nedeterministická prostředí [9]:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma \max_{\alpha} Q(s_{t+1}, a) - Q(s_t,a_t)] \tag{6}$$

kde:

α: $0 \leq \alpha \leq 1$ – rychlost učení. Nastavení na 0 znamená, že Q hodnoty nejsou nikdy aktualizovány a agent se nic neučí. Čím vyšší hodnota, tím vyšší rychlost učení.

$\max_{\alpha} Q(s_{t+1}, a)$ - je maximální odměna, již je možné získat v následujícím stavu – například odměna za následný výběr optimální akce [7].

Q-učení je speciální případ opakovaně posilovaného učení, při němž se agent místo funkce V učí akční hodnotovou funkci $Q(s, a)$ [10]. V tomto případě naučená akční funkce Q přímo aproximuje optimální akční funkci Q^* a může být dokázáno, že při nekonečném počtu opakování algoritmus vždy konverguje k optimální funkci Q^* s pravděpodobností 1 pro libovolnou strategii, dokonce i pro strategii naprosto náhodného výběru akcí. Obecný algoritmus Q-učení vypadá následovně [7]:

1 Inicializuj tabulku Q hodnot *Q(s,a)*
2 Opakuj pro každou epizodu
  2.1 Inicializuj počáteční stav *s*
  2.2 Opakuj dokud není *s* konečný stav
    2.2.1 Vyber akci *a* ze stavu *s* dle zvolené strategie (např. ε-hladová strategie)
    2.2.2 Proveď akci *a*, zjisti odměnu *r* a následující stav *s´*
    2.2.3 Aktualizuj *Q(s,a)* dle vztahu (1)
    2.2.4 $s \leftarrow s´$
3 Dokud neprojde stanovený počet epizod

Trénovací epizoda je jeden průchod prostředím, který končí buď dosažením cílového stavu nebo jiného terminálního stavu.

Protože algoritmus musí projít všechny stavy mnohokrát, aby mohl konvergovat k optimální funkci, proces učení může být velice zdlouhavý. Q-učení také může selhávat na prostředích s velice velkým množstvím stavů a nebo se skrytými či neúplnými stavy [10].

5 EXISTUJÍCÍ APLIKACE POSILOVANÉHO UČENÍ

Posilované učení lze při řešení problému plánování cesty využít více způsoby. Jedná se například o globální plánování, lokální plánování, nebo může být také použito pouze k upřesnění nebo optimalizaci výsledků jiné metody (případové usuzování, genetické algoritmy atd.).

Prvním krokem společným téměř pro všechny metody je určení reprezentace prostředí. Jelikož posilované učení je diskrétní metoda, je nutno provést transformaci spojitého reálného prostředí do diskrétního modelu. Pro globální plánováni cesty je vhodná aproximativní metoda rozkladu do buněk (kap. 3.2.2), pro lokální plánování cesty je nutné zvolit způsob diskretizace dle konkrétní metody. Je nutné si určit, jaký máme typ prostředí (známé/neznámé) a překážek (dynamické/statické), a dle toho zvolit plánovač a vhodnou metodu.

U globálních plánovačů nás zajímá především celková mapa prostředí, naše startovní poloha a poloha cíle. Určíme si odměny a postihy za přechod do dalšího neterminálního stavu, za přechod do cílového stavu a za přechod do nežádoucích stavů. Vhodná ukázka je například metoda popsaná v [2]. Následně agent prochází mapu prostředí, dokud nezíská optimální strategii pohybu daným prostředím. V závislosti na velikosti prostředí a zvolené metodě může být tento proces poměrně zdlouhavý, ale při dostatečném počtu průchodů konverguje k optimální strategii. Výkon globálních plánovačů je možné zlepšit pomocí heuristických metod. Globální plánovače jsou vhodné především pro známá prostředí se statickými překážkami.

U lokálních plánovačů je mnohem důležitější reprezentace okamžitého okolí agenta. Stav tak může být tvořen například relativní polohou cíle a relativní polohou překážek vůči agentovi (příklad viz [1]). Lokální plánovač také obvykle zahrnuje omezení pohybu robota (například robot se může pohybovat pouze dopředu, dopředu vlevo, dopředu vpravo a nemůže se otáčet na místě) a metodu pohybu k cíli – například nejpřímější možnou cestou, nebo dle určité cesty naplánované globálním plánovačem. Lokální plánování uvažuje plánování jako optimalizační problém, kde nalezení cesty souvisí s optimalizací nějaké dané funkce.

Lokální plánovače dokáží hledat nejvhodnější cestu mnohem rychleji a jsou schopny se pohybovat i v neznámém prostředí s dynamickými překážkami, nicméně především ve velice složitých prostředích (bludiště) se mohou často zaseknout v lokálních optimech a nenaleznout cestu k cíli.

V následujících kapitolách jsou shrnuty některé dosavadní aplikace posilovaného učení na konkrétní problém.

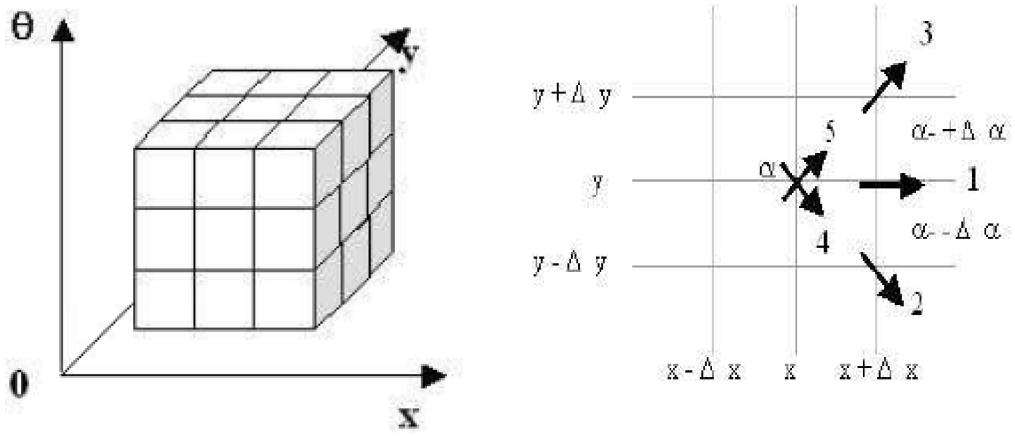
5.1 Plánování pohybu neholonomního robota ve statickém prostředí

Tato podkapitola popisuje řešení navržené v práci [2]. Cílem práce by měl být návrh plánovače schopného generovat cesty vhodné pro roboty s neholonomními omezeními.

5.1.1 Reprezentace prostředí

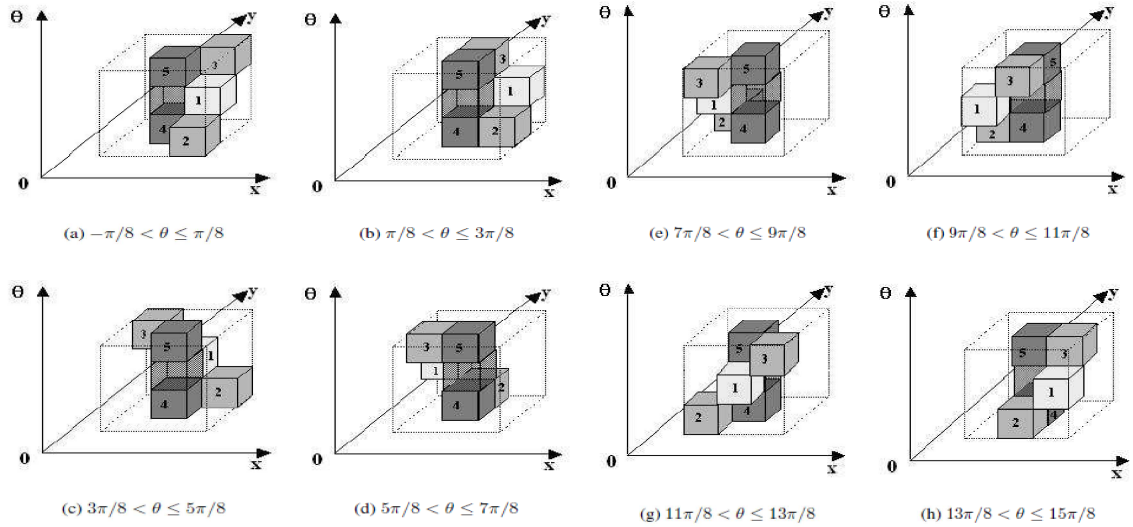
V práci je prezentován problém plánování pohybu neholonomního robota, který má k dispozici pět možných akcí – pohyb vpřed (1), vpravo vpřed (2), vlevo vpřed (3), rotace doprava (4) a rotace doleva (5). Akce jsou definovány relativně vzhledem k současné orientaci robota, následující akce je tedy závislá na současné orientaci.

$$K = (x,y,\theta) \tag{7}$$



Obr. 22 Stavový prostor a možné akce robota [2]

V zájmu zachování efektivity pro každý stav existuje maximálně osm následujících stavů, ačkoliv natočení robota může mít výrazně více kvantifikačních úrovní.



Obr. 23 Stavy následující po daných akcích v závislosti na orientaci robota [2]

5.1.2 Odměňovací funkce

V práci [2] je navržena jednoduchá ohodnocovací funkce určující odměny *r*.

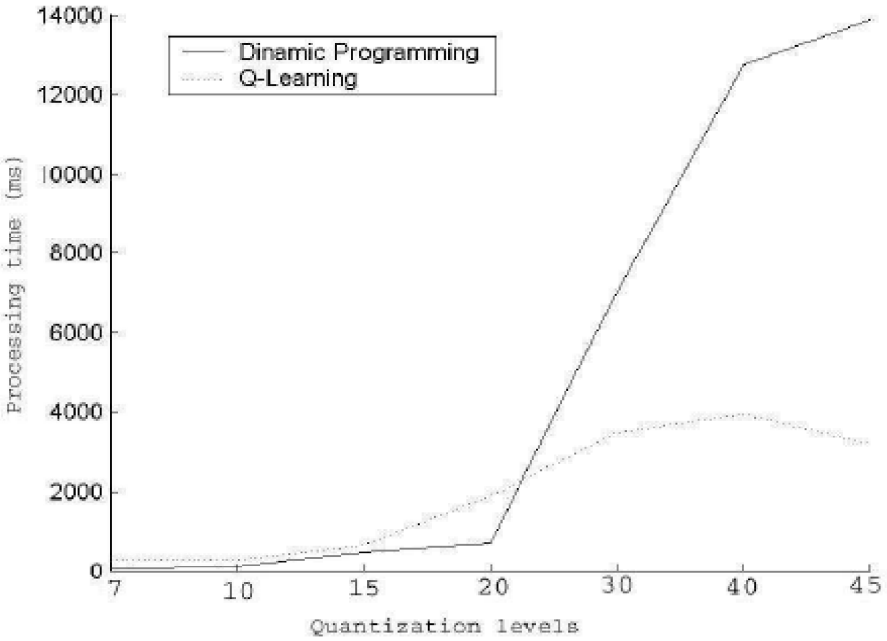
Popis funkce:

- odměna za volný stav $r = 0$
- odměna za stav mimo pracovní prostor $r = 0$
- odměna za finální stav $r = 1$
- odměna za stav obsahující překážku $r = -1$

Tato funkce by měla zajistit, že výsledná vygenerovaná cesta neobsahuje žádný překážkový stav. Při učení strategie je v případě kontaktu s překážkou nebo dosažení stavu mimo pracovní prostor následující stav vždy počáteční stav.

5.1.3 Volba učícího algoritmu

V práci jsou použity dva algoritmy posilovaného učení, jeden založený na dynamickém programování, druhý využívá Q-učení. Experimenty bylo zjištěno, že pro úlohy s méně než 8000 stavy je výhodnější použít dynamické programování, pro úlohy s počtem stavů mezi 8000 – 60000 je již výhodnější Q-učení. Pro více než 60 000 stavů autoři doporučují Q-učení zkombinovaný s nástrojem pro interpolaci hodnotové funkce, například neuronovou síť.



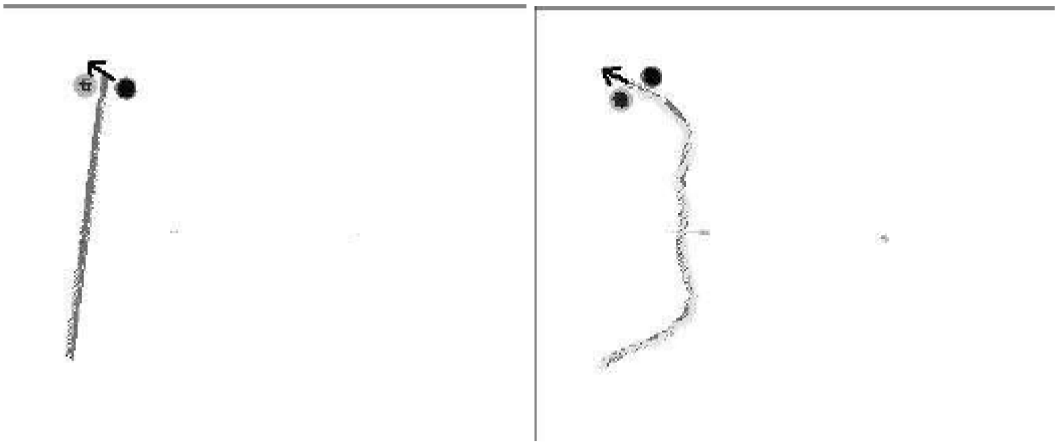
Obr. 24 Výpočetní čas v závislosti na kvantifikační úrovní pro oba algoritmy [2]

5.1.4 Výsledky

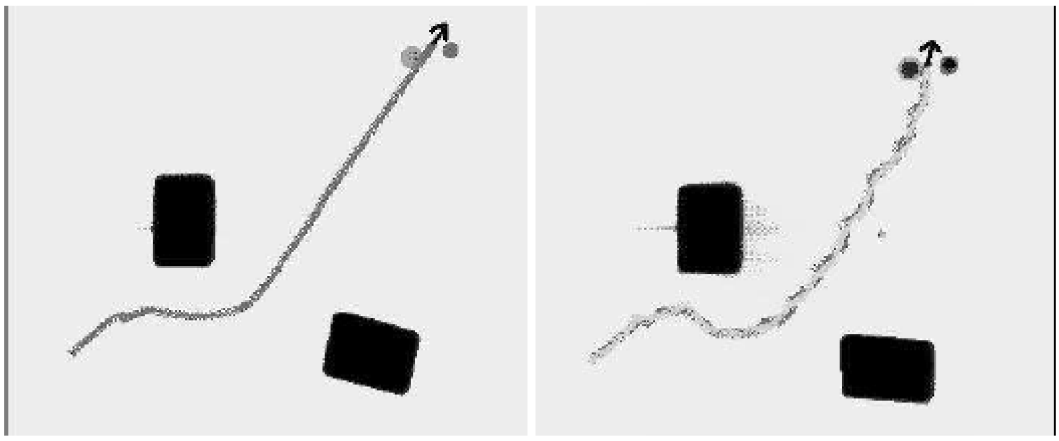
Autoři srovnávali výsledky nové metody především s depth-first prohledávacím algoritmem. Prezentované řešení bylo testováno na malém dvoukolovém robota.

Autoři robota vystavili pěti modelovým scénářům, s různými počátečními a cílovými konfiguracemi a různými počty překážek. Při experimentu bylo využito stavových prostorů s šesti různými kvantifikačními úrovněmi konfiguračního prostoru, kdy tři rozměry konfiguračního vektoru (x, y, θ) byly kvantifikovány hodnotami 7,10,15,20,30,45. Q-učení bylo otestováno pro různé množství iterací od 9000 do 70000. Výsledné hodnoty byly pravděpodobně získány ze souhrných výsledků těchto experimentů.

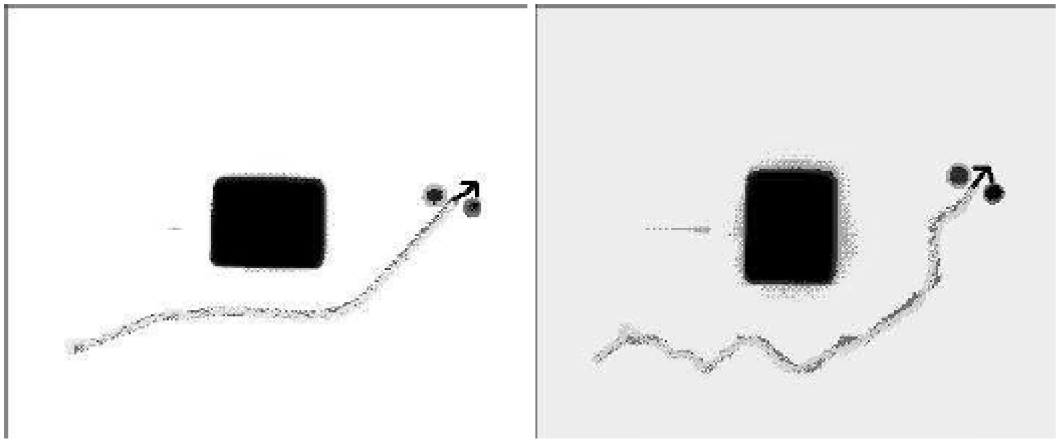
Na následujících obrázcích jsou prezentovány vygenerované trasy v různých modelových situacích pomocí obou srovnávaných algoritmů. Dvě kola na obrázku značí orientaci robota, šipka určuje požadovanou finální konfiguraci. Vygenerovaná cesta je vyznačena tmavě šedou, a reálná provedená cesta pak světle šedou barvou.



(a) Depth-First (b) Reinforcement Learning



(a) Depth-First (b) Reinforcement Learning



(a) Depth-First (b) Reinforcement Learning

Obr. 25 Vygenerované cesty pomocí posilovaného učení a depth-first algoritmu [2]

První ze sledovaných proměnných byly změny úhlů orientace. Tento výsledek je důležitý, protože určuje proveditelnost nalezené cesty robotem s neholonomickými omezeními.

Angle Variation	Reinforcement Learning	Depth-first
$0 \leq \Delta\theta < \pi/12$	88.39 %	96.93 %
$\pi/12 \leq \Delta\theta < \pi/6$	9.14 %	2.94 %
$\pi/6 \leq \Delta\theta < \pi/4$	1.67 %	0.00 %
$\pi/4 \leq \Delta\theta < \pi/3$	0.28 %	0.00 %
$\pi/3 \leq \Delta\theta < 5\pi/12$	0.40 %	0.00 %
$5\pi/12 \leq \Delta\theta < \pi/2$	0.03 %	0.00 %
$\pi/2 \leq \Delta\theta < 7\pi/12$	0.04 %	0.13 %
$7\pi/12 \leq \Delta\theta < 2\pi/3$	0.06 %	0.00 %

Obr. 26 Srovnání variace úhlů orientace [2]

Z výsledků je vidět, že depth-first má výrazně více nejmenších změn úhlů (v rozmezí $\langle 0 - \pi/12)$) než posilované učení, nicméně má nepatrně více velkých změn úhlů (v rozmezí $\langle \pi/2 - 7\pi/12)$). Autoři zde argumentují tím, že depth-first generuje přímější trajektorie, avšak s více náhlými změnami směru, zatímco posilované učení generuje zaoblenější trajektorie s méně ostrými křivkami.

Zde bych si nicméně dovolil nesouhlasit, protože z výše zobrazených trajektorií nic takového nevyplývá, naopak se cesty generované posilovaným učením zdají být mnohem klikatější a s více ostrými změnami orientace. Z obrázků je také vidět, že u posilovaného učení se vygenerovaná a provedená cesta neshodují, tudiž bylo pro robota pravděpodobně výrazně obtížnější se vygenerované cesty držet. Přesto je však navrhované řešení výrazně vhodnější pro neholonomní roboty, než většina klasických metod, generujících polygonální trasy.

Druhá z porovnávaných proměnných byla odchylka úhlu orientace od požadované finální konfigurace.

Final Orientation Error	Reinforcement Learning	Depth-first
$0 \leq \Delta\theta < \pi/12$	80.43 %	20.00 %
$\pi/12 \leq \Delta\theta < \pi/6$	13.04 %	20.00 %
$\pi/6 \leq \Delta\theta < \pi/4$	2.17 %	20.00 %
$\pi/4 \leq \Delta\theta < \pi/3$	0.00 %	0.00 %
$\pi/3 \leq \Delta\theta < 5\pi/12$	0.00 %	0.00 %
$5\pi/12 \leq \Delta\theta < \pi/2$	4.34 %	20.00 %
$\pi/2 \leq \Delta\theta < 7\pi/12$	0.00 %	0.00 %
$7\pi/12 \leq \Delta\theta < 2\pi/3$	0.00 %	20.00 %

Obr. 27 Srovnání odchylky orientace robota v cíli od požadované orientace [2]

Zde je jasně vidět výhoda posilovaného učení, které ve většině případů zaručuje velice malou odchylku od požadovaného úhlu, zatímco depth-first algoritmus nezaručuje vůbec nic.

Autoři také testovali rychlost výpočtu trasy, bohužel ji však nesrovnali s žádným alternativním přístupem, tudíž jsou tyto výsledky poměrně irelevantní.

5.2 Plánování pohybu v neznámém prostředí s pohyblivými překážkami

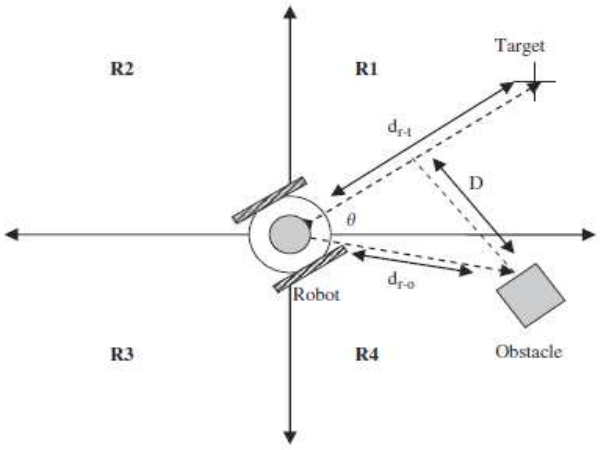
Využítím Q-učení při plánování cesty v neznámém prostředí s dynamickými překážkami se zabývá práce [1]. Cílem práce je navrhnout řešení, schopné plánovat cestu v dynamickém prostředí s několika dynamickými i statickými překážkami. Robot na začátku bude znát svou polohu i polohu cíle, a jeho úkolem je dostat se k cíli bez kolize s překážkou.

5.2.1 Předpoklady

1. Okolí je pro robota plně pozorovatelné, je tudíž schopen si pomocí sensorů obstarat všechny informace, jež bude potřebovat.
2. V každém okamžiku je známa rychlost a poloha robota, cíle, a všech překážek.
3. Rychlost robota je vždy vyšší než rychlost cíle a zároveň vyšší než rychlost dynamických překážek.
4. Tvar cíle i překážek jsou známy v každém časovém okamžiku.

5.2.2 Reprezentace prostředí

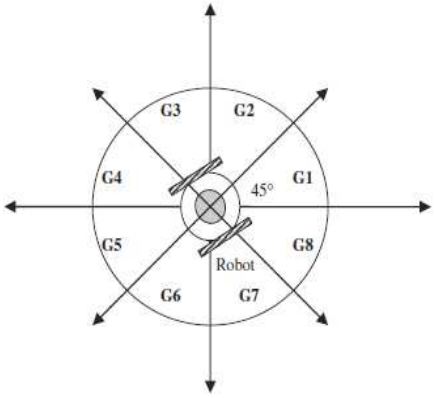
Prostředí je tvořeno překážkami a cílem. Jak překážky, tak cíl mohou být statické nebo dynamické. Robot je v každém časovém okamžiku přesným středem svého okolí, které je rozděleno do čtyř sektorů, *R1* až *R4*. Konfigurace robota je pak určena informací, ve kterém sektoru se nachází cíl, ve kterém sektoru nejbližší překážka a vzájemným úhlem θ mezi cílem a překážkou.



Obr. 28 Prostředí robota obsahující jednu překážku a jeden cíl [1]

Jelikož je úhel θ spojitá veličina, algoritmus by musel procházet nekonečné množství konfigurací, což je nežádoucí a navíc nadbytečné, protože algoritmu stačí znát přibližný úhel. Z důvodu redukce počtu možných konfigurací jsou tedy možné hodnoty úhlu θ rozděleny do osmi úhlových intervalů.

G1: $\theta \in \langle 0, \pi/4)$
G2: $\theta \in \langle \pi/4, \pi/2)$
G3: $\theta \in \langle \pi/2 , 3\pi/4)$
G4: $\theta \in \langle 3\pi/4 , \pi)$
G5: $\theta \in \langle \pi , 5\pi/4)$
G6: $\theta \in \langle 5\pi/4 , 3\pi/2)$
G7: $\theta \in \langle 3\pi/2 , 7\pi/4)$
G8: $\theta \in \langle 7\pi/4 , 2\pi)$



Obr. 29 Rozdělení úhlu do intervalů [1]

Tímto způsobem je tedy každá konfigurace přesně určena sektorem ve kterém leží cíl, sektorem ve kterém leží překážka a intervalem úhlu mezi překážkou a cílem:

$$S_t = (R_g, R_o, G_n), n \in \{1, ..., 8\} \quad (8)$$

Robot má neholonomní omezení a může vykonat tři akce: pohyb vpřed, otočení doleva, otočení doprava. Výsledek těchto akcí závisí na aktuální orientaci robota. Robot mění svou orientaci vždy přímo směrem k cíli a snaží se dosáhnout cíle nejkratší možnou cestou; akce otočení vlevo a otočení vpravo pak slouží k vyhýbání se překážkám. Úhel otočení vlevo/vpravo lze pak nastavit dle potřeb či požadavků.

5.2.3 Odměňovací funkce

V práci jsou rozlišeny 4 možné stavy:

- bezpečný stav (SS) – stavy s nízkou pravděpodobností kolize
- nebezpečný stav (NS) – stavy s vysokou pravděpodobností kolize
- cílový stav (WS) – stav, kdy robot dosáhl cíle
- kolizní stav (FS) – stav, kdy robot narazil do překážky

Bezpečný stav je obvykle určen minimální vzdáleností od nejbližší překážky. Pokud je tato vzdálenost menší, stav je považován za nebezpečný.

Odměňovací funkce pak má tuto podobu:

- odměna za přechod z NS do SS je r = 1
- odměna za přechod z SS do NS je r = –1
- odměna za přechod z NS do NS a zkrácení vzdálenosti k překážce je r = –1
- odměna za přechod z NS do NS a vzdálení se od překážky je r = 0
- odměna za dosažení WS je r = 2
- odměna za dosažení FS je r = –2

S využitím této funkce se robot snaží zůstávat v bezpečných oblastech a vyhýbat se překážkám, dokud nedosáhne finálního stavu.

5.2.4 Ohodnocovací funkce

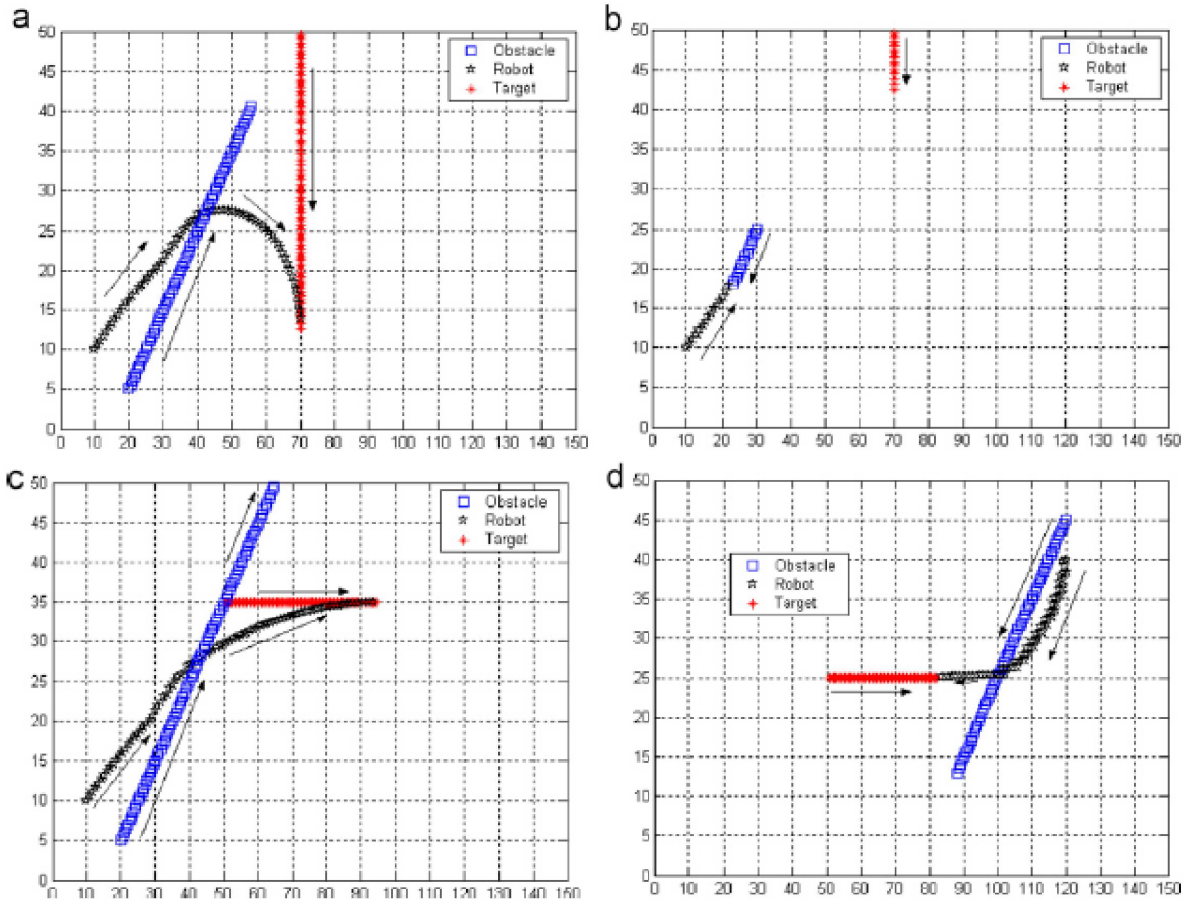
K uchování Q-hodnot je využita dynamicky tvořená tabulka, ve které jsou řádky

zastoupeny jednotlivými stavy a sloupce jsou dané dostupnými akcemi. Q-hodnoty jsou vypočítávány dle vztahu pro deterministické Q-učení (5).

Tato tabulka je před tréninkem inicializována na nulové hodnoty, a následně iteračně aktualizována během tréninku. Získaná strategie je poté používána robotem při navigování v reálném provozu, konkrétně ve chvíli, kdy se dostane do nebezpečného stavu. Pokud není v nebezpečném stavu, pokračuje nejkratší cestou k cíli. Pokud je v nebezpečném stavu, identifikuje současný stav v tabulce a vybere akci s nejvyšší Q hodnotou.

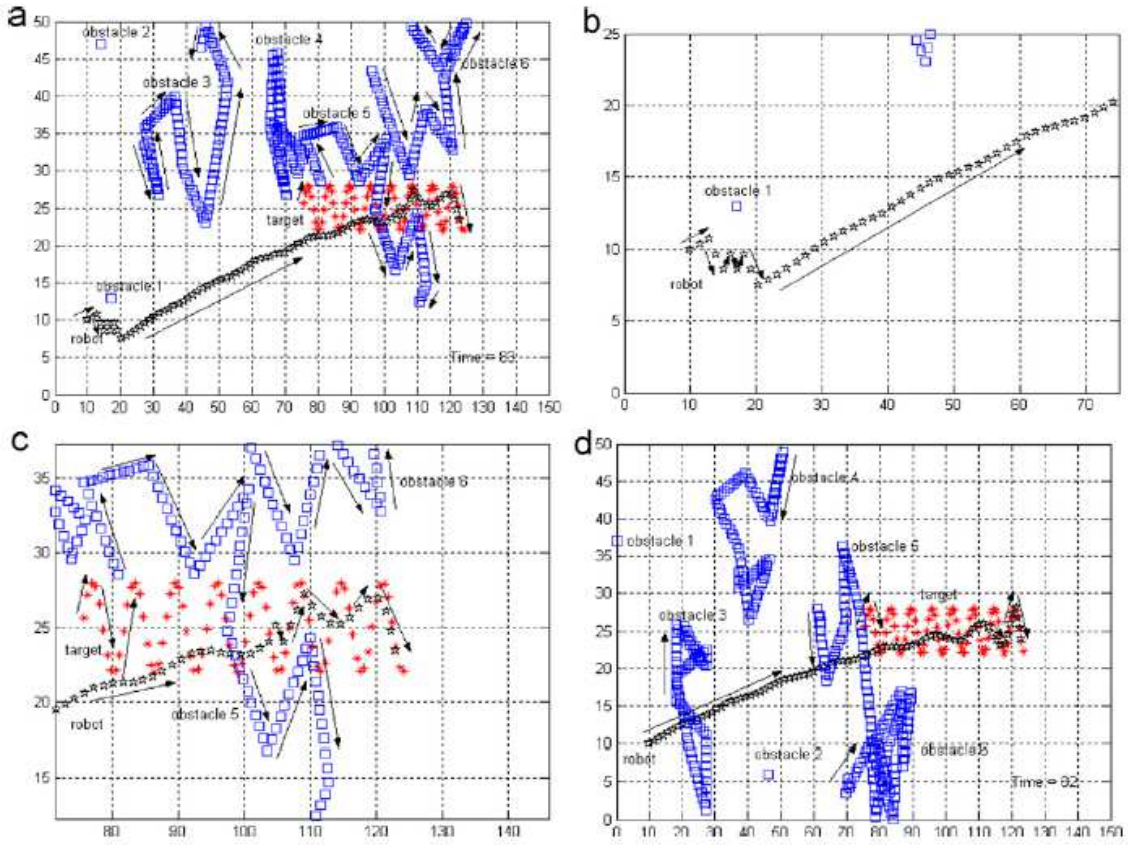
5.2.5 Simulace

Simulace vždy probíhaly ve dvou fázích: trénovací fázi a testovací fázi. V trénovací fázi byl agent vystaven více trénovacím scénářům; v každém z nich měla překážka i cíl jinou počáteční polohu a směr pohybu.



Obr. 30 Trénovací scénáře [1]

V testovací fázi je následně robot vystaven výrazně složitějším scénářům, aby se ověřila funkčnost navrženého řešení. Tyto složitější scénáře zahrnují více překážek, z nichž některé jsou statické a jiné dynamické. Scénáře jsou buď náhodně generované, nebo navržené speciálně za účelem testování určitých vlastností navrženého řešení (například testování problému lokálního minima). Pro pohyb překážek byl zvolen algoritmus náhodné chůze, aby se dosáhlo realistických situací. Za zmínku také stojí, že testovací scénáře zároveň slouží i jako trénovací, protože robot si při jejich průchodu stále aktualizuje svou tabulku Q hodnot.



Obr. 31 Testovací scénáře

5.2.6 Výsledky simulací

Nejdůležitějším parametrem plánovače je míra úspěšnosti, tedy kolikrát se robot dokázal dostat do finálního stavu a kolikrát jeho pokus skončil kolizí. Tato úspěšnost je přímo závislá na kvalitě i kvantitě tréninku a na počtu překážek v prostředí. Očekávaný výsledek je, že čím více tréninku a méně překážek, tím vyšší úspěšnost.

Při testování úspěšnosti byly zvoleny scénáře, ve kterých byly náhodně rozmístěné statické překážky, pohyblivé překážky využívající random walking model a cíl pohybující se po sinusoidě.

První test hledal vztah mezi počtem tréninků a úspěšností. Robot byl trénován na určitém počtu trénovacích scénářů a následně procházel 500 náhodných testovacích prostředí. Výsledky zaznamenává následující obrázek.

# of Training scenarios	#of Hits out of 500	Miss rate (%)
5	409	18.2
15	417	16.6
30	429	14.2
45	436	12.8
65	481	3.8
75	490	2
90	421	15.8
100	412	17.6

Obr. 32 Míra úspěšnosti v závislosti na tréninku [1]

Výsledky dokazují, že předpoklad vyšší úspěšnosti s počtem trénovacích scénářů je částečně správný, nicméně od 75 tréninků výše dochází k takzvanému „přeučení“. To způsobuje, že se systém soustředí na pár specifických případů a úspěšnost rapidně klesá.

Druhý test hledal vztah mezi mírou úspěšnosti a počtem překážek ve scénáři. Při tomto testu byl robot trénován 50x a následně testován opět na 500 scénářích. Na počátku byly ve scénáři dvě dynamické překážky a jedna statická, a při dalších testech byla vždy přídána jedna statická a jedna dynamická překážka.

# of obstacles	#of Hits out of 500	Miss rate (%)
3	490	2
5	488	2.4
7	484	3.2
9	483	3.4
11	407	16.28
13	378	24.4

Obr. 33 Míra úspěšnosti v závislosti na počtu překážek [1]

Z výsledků je jasně vidět, že systém má velice vysokou úspěšnost, pokud je počet překážek nižší nebo roven 9. Pro vyšší počty překážek se již úspěšnost rapidně snižuje. Pravděpodobným důvodem je způsob určování konfigurace robota, jelikož bere v úvahu pouze nejbližší překážku i ve chvíli kdy může být nebezpečně blízko dvěma a více překážkám.

6 NÁVRH A IMPLEMENTACE VLASTNÍCH ALGORITMŮ

Ve své práci jsem se rozhodl implementovat tři různé algoritmy založené především na Q-učení. Jelikož nebyly specifikovány parametry robota a prostředí, algoritmy jsem vybral a implementoval dle svého vlastního uvážení – vytvořil jsem tedy dva algoritmy pro globální plánování a jeden algoritmus pro lokální plánování.

Algoritmy mohou být vhodné pro různé typy prostředí a pro různé situace, z čehož plynou jejich jednotlivé výhody a nevýhody. Algoritmy byly uzpůsobeny tak, aby je bylo možné použít ve stejném grafickém prostředí, konkrétně diskrétní dvourozměrné mapě. To vedlo k několika menším kompromisům, které způsobily, že algoritmy jsou vhodné především pro demonstraci možností Q-učení. Návrh algoritmů je inspirován primárně pracemi [1],[2] a dále demonstračními implementacemi volně dohledatelnými na webu, např. [14],[15],[16],[17].

Mým hlavním cílem bylo ukázat univerzálnost posilovaného učení pro řešení úloh s různými požadavky či omezeními a ověřit jejich efektivitu v různých situacích.

6.1 Reprezentace testovaných prostředí

Mapa reprezentuje prostředí, ve kterém se robot může pohybovat. Grafické prostředí je tvořeno čtvercovou dvourozměrnou mřížkou s volitelným počtem buněk, vnitřní interpretace je dvourozměrný souřadnicový systém [*x*,*y*], kdy je každá souřadnice zároveň parametrem stavu robota ve stavovém prostoru. Množství buněk (možných stavů) pak zvyšuje maximální možnou přesnost algoritmů, zároveň však většinou zvyšuje jejich výpočetní náročnost.

0,0	1,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0	9,0
0,1	1,1	2,1	3,1	4,1	5,1	6,1	7,1	8,1	9,1
0,2	1,2	2,2	3,2	4,2	5,2	6,2	7,2	8,2	9,2
0,3	1,3	2,3	3,3	4,3	5,3	6,3	7,3	8,3	9,3
0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4
0,5	1,5	2,5	3,5	4,5	5,5	6,5	7,5	8,5	9,5
0,6	1,6	2,6	3,6	4,6	5,6	6,6	7,6	8,6	9,6
0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	8,7	9,7
0,8	1,8	2,8	3,8	4,8	5,8	6,8	7,8	8,8	9,8
0,9	1,9	2,9	3,9	4,9	5,9	6,9	7,9	8,9	9,9

Obr. 34 Mapa prostředí 10x10 s označením [x,y] souřadnic

Pohyb po mapě je obecně možný ve všech osmi směřech. Pro potřeby algoritmů jsou jednotlivé možné akce očíslovány a určeny absolutně, tj. bez ohledu na aktuální orientaci robota. Pokud je tedy robot na pozici např. [1,3] a provede akci (7), jeho následující pozice bude vždy [2,4]. Pro každou pozici je specifikován vlastní seznam akcí, které jsou pro ni dostupné, například na pozici [0,2] jsou akce (0), (3), (5) nepřístupné, jelikož by vedly mimo stavový prostor. Také v případě plánování cesty pro robota s omezením pohybu robotu povolíme pouze některé stavy.

0	1	2
3	-	4
5	6	7

Obr. 35 Číslování akcí

V každém stavu se uchovávají hodnoty potřebné pro běh všech tří algoritmů. Tato vlastnost sice snižuje paměťovou efektivitu, ale umožňuje spouštět všechny tři algoritmy a vykreslovat cesty jimi vygenerované zároveň, což je vhodné pro demonstrační účely.

Jednotlivé pozice na mapě mohou být ve 4 stavech:
- pozice je prázdná – volný stav
- pozice je neprostupná – překážkový stav
- pozice je cílová – cílový stav
- pozice je startovací – startovní stav

6.2 Globální Q-učení ve statickém prostředí bez omezení i s omezením pohybu

Algoritmus dále označovaný jako GQ, předpokládá libovolně uspořádané statické prostředí. Tento algoritmus byl navržen pro plánování pohybu všesměrového robota, a následně mírně upraven pro plánování cesty robota s neholonomním omezením. V tom případě se robot může pohybovat pouze ve směru dopředu, dopředu vlevo a dopředu vpravo v závislosti na současné orientaci.

6.2.1 Reprezentace stavu

Stav robota je jednoznačně určen jeho pozicí v souřadnicovém systému. Pro každý stav má robot k dispozici sadu proměnných, jež využívá k dalšímu plánování. Jedná se o:
- seznam aktuálně dostupných akcí v tomto stavu
- seznam Q-hodnot odpovídajících dostupným akcím
- pomocnou proměnnou určující, zda je na pozici překážka
- odměnu za dosažení tohoto stavu
- hodnotu zaznamenávající, kolikrát byl stav již navštívěn během tréninku

6.2.2 Odměňovací a ohodnocovací funkce

Na počátku jsou Q-hodnoty všech stavů inicializovány na nulu. Ohodnocování pak probíhá dle vztahu (6). Pokud je provedená akce diagonální, tak výslednou Q-hodnotu vynásobíme koeficientem menším než jedna, protože je nutné zohlednit delší cestu při pohybu diagonálně.

Odměna za akci je určena stavem následujícím po aktuálně provedené akci. Odměňovací funkce je určena takto:

- za volné stavy je odměna r = 0
- za cílový stav je odměna r > 0
- za překážkový stav je odměna (trest) r < 0

Pro obecný případ doporučuji aby absolutní hodnoty odměn a trestů byly přibližně stejné. Takto nastavené odměny/tresty zajišťují, že se agent bude vyhýbat překážkám a zároveň se snažit dosáhnout cíle nejkratší možnou cestou.

6.2.3 Strategie výběru akcí

Pro výběr akcí byla implementována mirně upravená ε-greedy strategie. Tato strategie vybírá ve většině případů dostupnou akci s nejvyšší Q-hodnotou, ale v určitém procentu případů (určeném parametrem ε) vybírá naprosto náhodně ze všech aktuálně dostupných akcí.

Pokud jsou všechny Q-hodnoty pro daný stav nulové nebo menší, algoritmus vybírá okolní stavy, které byly zatím nejméněkrát navštíveny. Tento způsob se ukázal efektivnější než čistě náhodný výběr, především pak pro rozsáhlejší stavové prostory.

Jelikož neexistuje metoda pro obecně optimální nastavení parametru ε, je nutné tento parametr určit odhadem či experimentálně. Pří hodnotě $\varepsilon = 0$ algoritmus hledá cestu velice rychle, tato cesta však až na vyjímky není optimální, naopak při $\varepsilon = 1$ je cesta optimální nebo jí blízká, nicméně výpočet trvá obvykle výrazně déle. Pro většinu typů prostředí se osvědčily hodnoty ε v intervalu $\langle 0{,}2;0{,}5\rangle$.

6.2.4 Učení mapy

Učení strategie pohybu pro dané prostředí probíhá dle následujícího mirně upraveného algoritmu Q-učení:

1 Opakuj pro každou epizodu
- 1.1 Inicializuj *s´* jako startovní stav
- 1.2 Opakuj pro každý krok epizody
  - 1.2.1 $s \leftarrow s'$
  - 1.2.2 Pomocí ε-greedy vyber akci *a* z akcí dostupných pro stav *s*
  - 1.2.3 Zjisti odměnu *r* a maximální Q-hodnotu pro následující stav *s´*
  - 1.2.4 Aktualizuj *Q(s,a)* dle vztahu (6)
  - 1.2.5 Jestliže je následující stav překážka nebo z následujícího stavu nevedou žádné další akce:
    - 1.2.5.1 Odeber provedenou akci *a* ze seznamu akcí dostupných pro stav *s*
    - 1.2.5.2 $s' \leftarrow s$ //Zůstane v současném stavu
- 1.3 Dokud *s* není finální stav nebo počet kroků nepřesáhne určitou hranici

2 Dokud nedosáhneš stanoveného počtu epizod

Algoritmus se v pár detailech liší od učícího algoritmu použitého např. v práci [2]. Především pokud je následující stav překážkový:

- neukončujeme epizodu a nezačínáme opět ve startovním stavu, ale pouze vybereme jinou dostupnou akci.
- akci, která vedla do překážkového stavu odstraníme ze seznamu dostupných akcí, takže ji agent v tomto stavu již nikdy nevykoná.

Tyto úpravy se při pokusech kladně projevily na rychlosti hledání a kvalitě hledané cesty při stejném počtu tréninkových epizod, především pak ve velice složitých prostředích.

6.2.5 Generování cesty

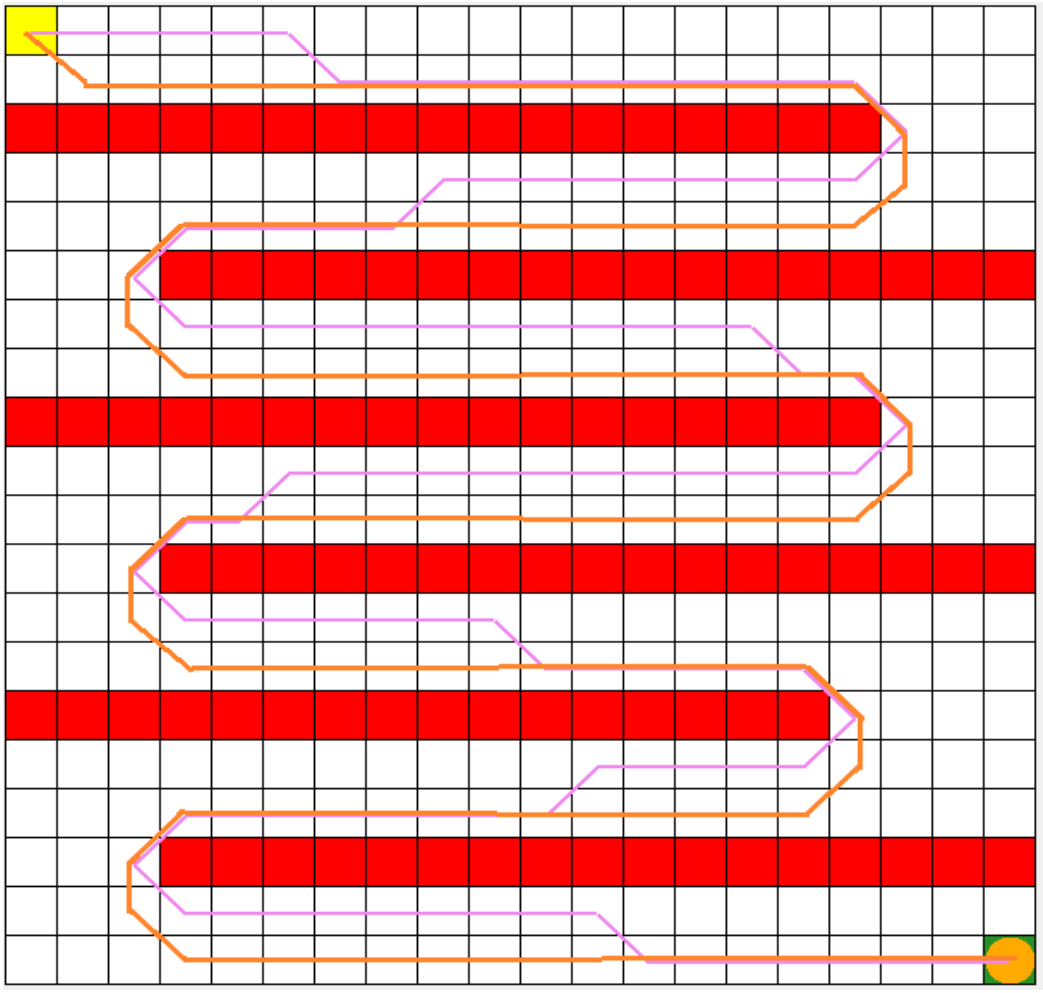
Pro generování cesty stačí postavit agenta do počáteční pozice, a za pomocí greedy strategie vybírat nejlepší akci pro daný stav.

Mezi užitečné výhody plánování cesty pomocí Q-učení patří schopnost najít cestu do cíle, i když robota mírně vychýlíme z jeho trasy. Při dostatečném prozkoumávání a dostatečném počtu iterací během učení si algoritmus vytvoří strategii pohybu i pro stavy, které pro nejkratší cestu nepotřebuje. Pokud by tedy robot byl při své cestě vychýlen, je ve většině případů schopen pokračovat dál k cíli.

6.2.6 Plánováni cesty s omezením pohybu

Algoritmus Q-učení může být snadno použit i pro plánování pohybu s určitými omezeními. V tomto případě má robot v každém stavu k dispozici vždy maximálně tři akce, závislé na svém aktuálním natočení, tedy na předchozí vykonané akci.

Pokud robotova předchozí akce byla např. (4) (směr vpravo), je i jeho současné natočení směrem (4), a zpřístupníme mu pouze akce (2), (4), (7), tedy z jeho pohledu dopředu vlevo, dopředu a dopředu vpravo. Stejně tak při hledání maximální Q-hodnoty z následujícího stavu (viz vztah (1)), uvažujeme pouze Q-hodnoty těch akcí, jež by robot mohl vykonat v závislosti na své budoucí orientaci. Srovnání výsledných cest viz obrázek. Počáteční natočení robota bylo směrem dolů (6).



Obr. 36 Naplánované cesty algoritmem Q-učení.
Fialová cesta – bez omezení pohybu. Oranžová cesta – s omezením pohybu

6.3 Zjednodušené posilované učení pro statické prostředí

Algoritmus klasického Q-učení není příliš vhodný pro triviální plánování cesty všesměrového robota ve statickém prostředí. Je totiž potřeba velké množství iterací ke konvergenci algoritmu, což výrazně zvyšuje výpočetní náročnost, především pak při větším množství stavů. Pokud tedy není k použití Q-učení dobrý důvod (například právě robot s neholonomními omezeními nebo požadavek co nejmenšího počtu změn směru, viz práce [18]), klasické výpočetní metody jsou obvykle efektivnější.

Pokusil jsem se tedy vytvořit algoritmus (dále označovaný RLL), který by byl výrazně jednodušší a rychlejší, ale zároveň by si zachovával některé dobré vlastnosti Q-učení, např. schopnost mapování a plánování cesty i v neznámém prostoru, a především schopnost plánování cesty z jakéhokoliv místa na mapě, čímž by se zároveň odlišoval od klasických algoritmů hledání nejkratší cesty, např. Dijkstrova algoritmu [19].

U navrženého algoritmu ohodnocujeme pouze kvalitu stavu (tzn. „jak dobré je na této pozici být“), nikoliv akce vedoucí ze stavu nebo do stavu. Každý stav je při vhodném způsobu ohodnocování nutno projít pouze jednou, což dohromady výrazně snižuje výpočetní i paměťovou náročnost.

6.3.1 Reprezentace stavu

Stav je vždy jednoznačně určen svými souřadnicemi ve stavovém prostoru. Pro každý stav jsou dány následující proměnné:

- odměna *r* za dosažení konkrétního stavu
- funkční hodnota *V* pro posouzení kvality stavu
- pomocná proměnná udávající, zda je stav již aktualizován
- pomocná proměnná určující, zda je stav překážka

6.3.2 Ohodnocovaní stavů

Pro určení kvality stavu je využíván následující vzorec:

$$V(s) = r(s) + \gamma \max_{s'} V(s') \tag{9}$$

- $r(s)$ je odměna za dosažení aktuálního stavu
- s' jsou sousední stavy

Pokud má nejvyšší funkční hodnotu $V(s')$ stav nacházející se diagonálně od současného stavu, pak výslednou funkční hodnotu $V(s)$ vynásobíme koeficientem menším než jedna.

Odměny jsou určeny stejným způsobem jako v předchozím algoritmu tj.

- za volné stavy je odměna $r = 0$
- za překážkové stavy je odměna (trest) $r < 0$
- za cílový stav je odměna $r > 0$

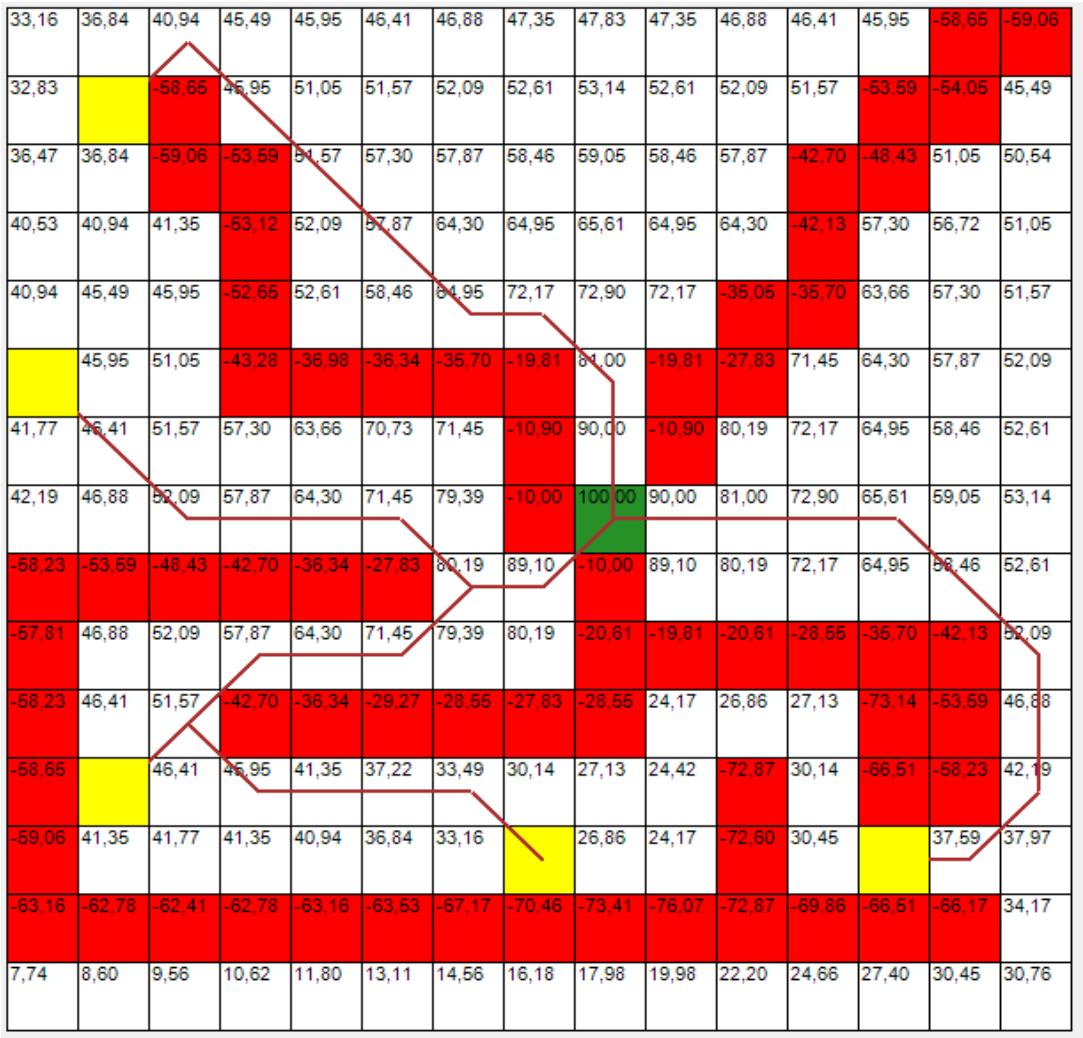
Konkrétní hodnoty pak může zadat přímo uživatel před spuštěním algoritmu.

6.3.3 Strategie ohodnocování

Aby bylo prohledávání prostoru co nejefektivnější, stavy jsou ohodnocovány směrem od cíle, dokud nejsou ohodnoceny všechny stavy ve stavovém prostoru.

1. Inicializuj prázdný seznam stavů k expanzi *open*
2. Inicializuj cílový stav $s_{cíl}$, nastav $V_{cíl} = r_{cíl}$, označ $s_{cíl}$ jako aktualizovaný, přidej $s_{cíl}$ do seznamu *open*
3. Opakuj, dokud nejsou všechny stavy *s* aktualizovány
   - 3.1 Vyber stav *s* na začátku seznamu *open* a vyjmi jej ze seznamu
   - 3.2 Opakuj pro všechny sousední stavy *s´*
     - 3.2.1 Jestliže *s´* ještě není aktualizovaný
       - 3.2.1.1 Ohodnoť stav *s´* podle vztahu (9)
       - 3.2.1.2 Pokud stav *s´* není překážkový, přidej jej na konec seznamu *open*

Pro následné generování cesty agent začíná na libovolné startovní pozici. Při pohybu vždy zkontroluje sousední stavy, vybere z nich nejkvalitnější stav (s nejvyšší funkční hodnotou *V*) a přesune se na něj. Tento postup opakuje dokud nedosáhne cílového stavu.



Obr. 37 Demonstrace schopnosti nalezení cesty do cíle z jakékoliv startovní lokace. Číselné hodnoty reprezentují kvalitu stavu

6.4 Lokální Q-učení v neznámém statickém prostředí

Oba výše uvedené algoritmy jsou globální plánovače navržené pro práci ve statickém prostředí. Strategie pohybu se tedy učí pouze pro dané prostředí a nemusí již dále platit při změně jakékoliv z překážek, což snižuje praktickou využitelnost takovýchto řešení. Navíc pro řešení poměrně triviálního problému globálního plánování cesty ve známém prostředí je Q-učení až příliš komplexní nástroj, protože jak již bylo naznačeno, na tento problém často stačí klasické výpočetní postupy.

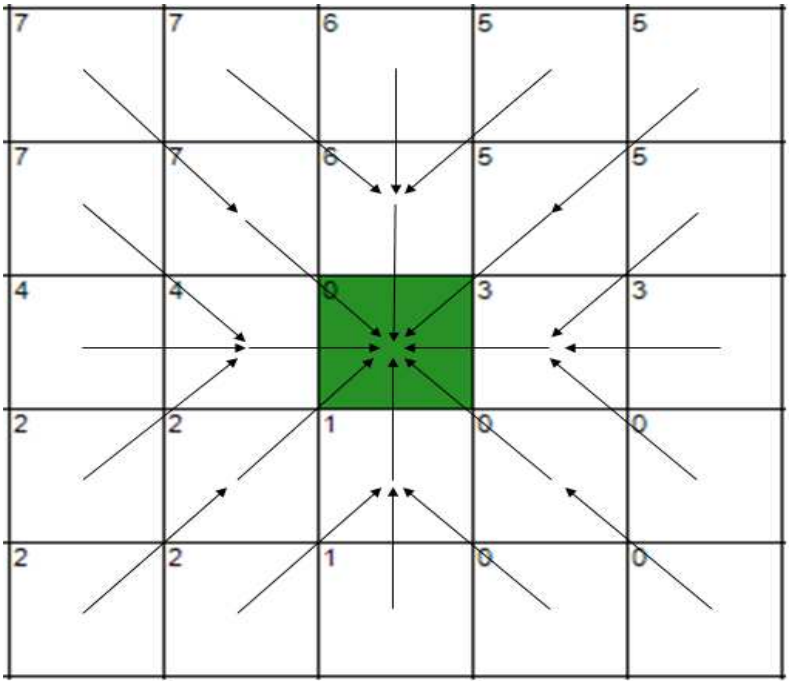
Mnohem zajímavější způsob využití pro Q-učení jsou lokální plánovače cesty. Ty jsou schopné úspěšně navigovat robota přes naprosto neznámé prostředí, bez potřeby se jej zdlouhavě učit metodou pokus-omyl, a to s využitím obecné strategie pohybu. Taková strategie není vázaná na konkrétní prostředí a jeho stavy, ale spíše na jednotlivé situace a jejich podobnost. V principu se robot snaží identifikovat svůj současný stav, podívat se jestli v takovém stavu již někdy byl a na základě svých dřívějších rozhodnutí a jejích výsledků zvolit nejlepší možnou akci. Tento algoritmus je dále označován jako LQ.

6.4.1 Reprezentace prostředí

U lokálních plánovačů navržených pro neznámá nebo dynamická prostředí tedy nelze stav identifikovat podle jeho souřadnic, protože poloha překážek vůči robotu může být vždy odlišná, přestože je na stejné pozici. Je tedy nutné zvolit jiný, obecnější způsob reprezentace stavů. V navrženém řešení je konfigurace robota určena počtem a relativní polohou překážek v nejbližším okolí, a polohou cíle vzhledem k současnému stavu.

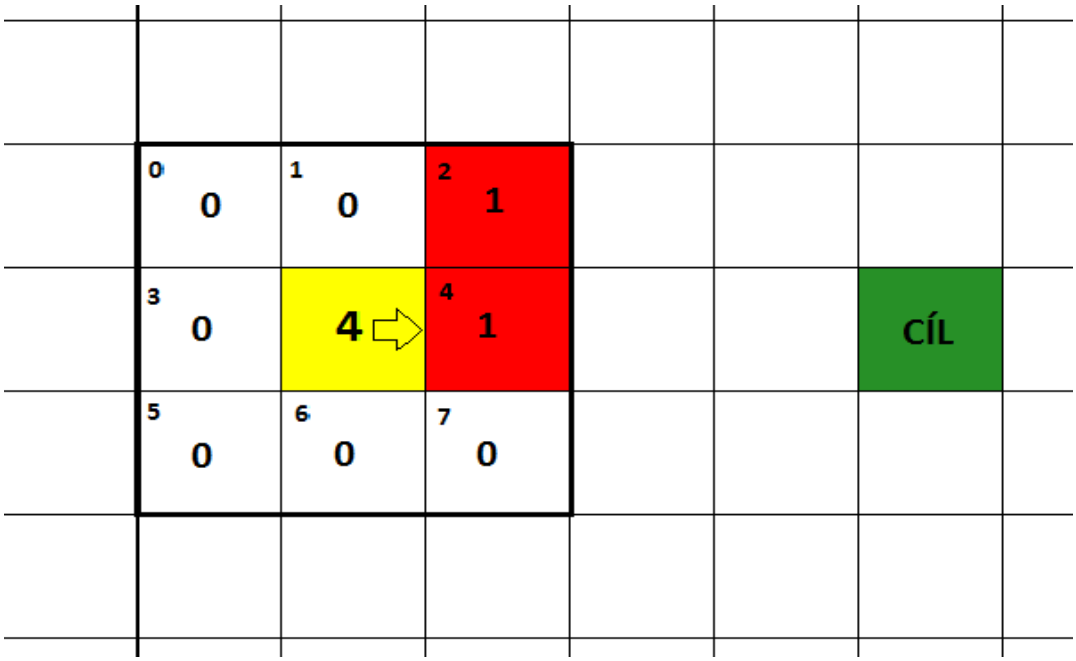
Každý stav na mapě má až osm dalších sousedních stavů, a na každém z nich může nebo nemusí být překážka. Zda je stav překážka lze vyjádřit binární hodnotou, a seznam osmi těchto binárních hodnot tvoří základ určování aktuálního stavu robota.

Protože jsou polohy sousedních stavů a do nich vedoucích akcí určeny absolutně, je nutno tuto reprezentaci obohatit o informaci obsahující aktuální relativní polohu cíle. Tu vyjádříme číslem akce, která vede do následujícího stavu nejblížšího k cíli. Hodnota relativní polohy cíle tak může nabývat osmi různých možností, viz obr. 38.



Obr. 38 Ukázka reprezentace relativní polohy cíle v různých stavech

Aktuální stav je tedy určena devíti hodnotami, osm z nich vyjadřuje polohu překážek relativně k současné poloze robota, a devátá určuje směr k cíli.



Obr. 39 Vizualizace aktuálního stavu robota

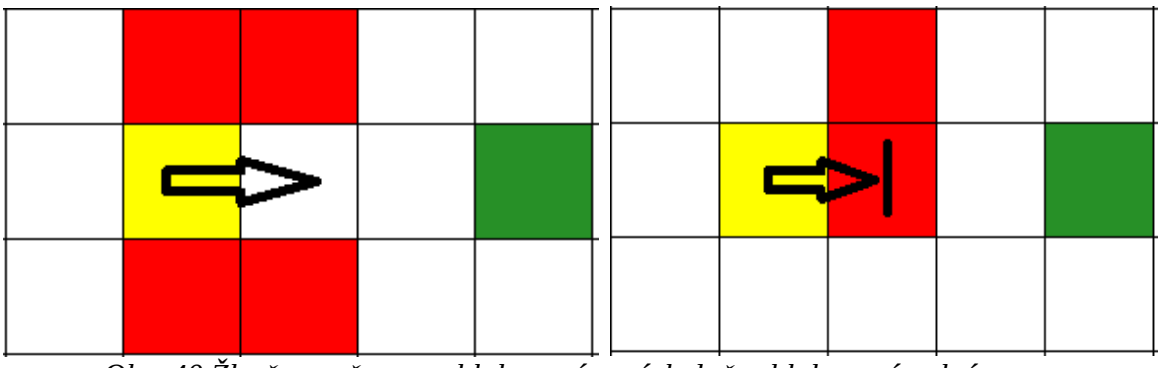
Žluté pole značí aktuální pozici robota, číslo a šipka značí směr, ve kterém leží cíl. Okolní hodnoty určují, zda je stav překážka, a tyto hodnoty dohromady tedy určují stav, na který robot bude reagovat.

Tento model tedy předpokládá robota vybaveného detekčními senzory ve všech osmi směrech a znalost polohy cíle vzhledem ke své vlastní poloze. Díky tomu je schopen rozeznávat jednotlivé konfigurace, ve kterých se nachází tak, aby na ně mohl adekvátně reagovat.

6.4.2 Odměňovací funkce

Při řešení plánování cesty rozlišujeme čtyři základní druhy stavů.

- Překážkový stav – stav kdy robot narazil do překážky
- Volný nezablokovaný stav – stav kdy na pozici vedoucí přímo k cíli neleží překážka
- Volný zablokovaný stav – stav kdy na pozici vedoucí přímo k cíli leží překážka
- Cílový stav



Obr. 40 Žlutě označen nezablokovaný a následně zablokovaný volný stav

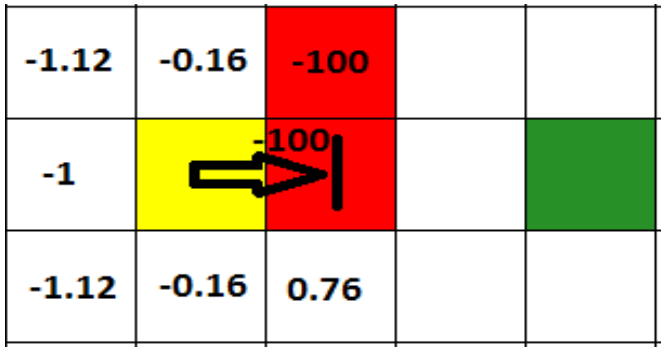
Pokud jsme v nezablokovaném stavu, tak vybíráme akci, vedoucí nejkratší cestou přímo k cíli. Pokud však narazíme na zablokovaný stav, je nutné vybrat jinou akci. Pro určení této akce je třeba zavést odměňovací funkci.

V implementovaném algoritmu jsou vytvořeny dvě varianty poměrně jednoduché odměňovací funkce, přičemž každá způsobuje poněkud jiné chování algoritmu.

Varianta 1

- Pokud je následující stav překážka – odměna $r = -100$
- Pokud je následující stav volný – odměna r je definována jako vzdálenost, o kterou se agent dostane blíže k cíli (rozdíl vzdálenosti současného stavu s_t a vzdálenosti následujícího stavu s_{t+1} od cíle).

Tato varianta ohodnocuje každou akci hodnotou, která určuje jak moc akce přiblíží agenta směrem k cíli. Agent se tak snaží vždy vybírat akce vedoucí k cíli nejkratší možnou cestou okolo nejbližších překážek.

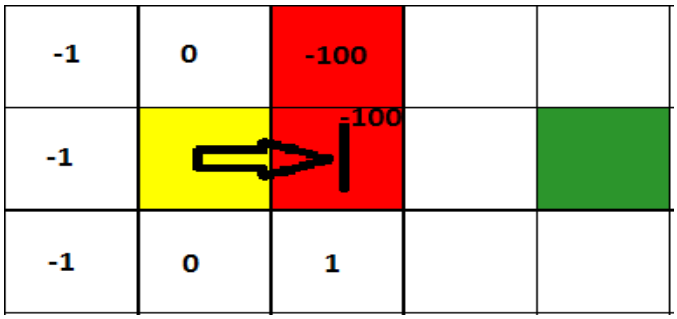


Obr. 41 Odměny za různé akce v zablokovaném stavu 1. varianta

Varianta 2

- Následující stav je překážka – odměna $r = -100$
- Následující stav je nezablokovaný a blíže k cíli než současný, pak $r = 1$
- Následující stav je nezablokovaný a stejně daleko nebo dále k cíli, pak $r = -1$
- Pro všechny ostatní stavy $r = 0$

U této varianty se agent důsledně vyhýbá všem překážkám, a snaží se najít akce, které s největší pravděpodobností vedou do nezablokovaného stavu, a tudíž co nejrychleji k cíli.



Obr. 42 Odměny za různé akce v zablokovaném stavu 2. varianta

6.4.3 Trénovací algoritmus

Trénink strategie pohybu začíná vytvořením trénovacího scénaře, tzn. vygenerováním překážek a vložením startu a cíle. Tento scénář je následně procházen v mnoha trénovacích epizodách, kdy jsou identifikovány jednotlivé konfigurace a vypočítávány Q-hodnoty pro různé akce v různých stavech v závislosti na odměňovací funkci.

Strategie pohybu je implementována jako globální seznam stavů *Qpolicy*, obsahující všechny identifikované stavy, které algoritmus během tréninku všech scénářů navštívil. Pro každý z těchto stavů jsou určeny dostupné akce a jejich Q-hodnoty, se kterými algoritmus dále pracuje.

Algoritmus pro trénink jednoho scénáře:

1 Opakuj pro každou epizodu
- 1.1 Inicializuj *s´* jako startovní stav
- 1.2 Opakuj pro každý krok epizody
  - 1.2.1 $s \leftarrow s´$
  - 1.2.2 Jestliže je stav *s* zablokovaný
    - 1.2.2.1 Najdi současný stav *s* v *Qpolicy* - jestliže nebyl nalezen, přidej jej do seznamu.
    - 1.2.2.2 Pomocí klasické ε-greedy strategie vyber nejvhodnější akci *a*
    - 1.2.2.3 Najdi nasledující stav *s´* v *Qpolicy* - jestliže nebyl nalezen, přidej jej do seznamu
    - 1.2.2.4 Aktualizuj *Q(s,a)* podle vztahu (6)
  - 1.2.3 Jestliže stav *s* není zablokovaný
    - 1.2.3.1 Vyber akci *a* vedoucí do stavu *s´* //směřující přímo k cíli
- 1.3 Dokud není *s* finální stav nebo počet kroků nepřesáhne určitou hranici

V implementovaném řešení je vytvořena metoda pro trénink všeobecně použitelné strategie. Tato metoda si vytváří vlastni trénovací mapu o rozměrech 20x20, náhodně vygeneruje překážky, vloží cílový stav doprostřed a startovací stav náhodně ke kraji mapy. Po stanoveném počtu epizod se stejným způsobem vytvoří nový scénář a učení pokračuje. Při dostatečném počtu scénářů by robot na konci měl mít obecně použitelnou strategii pohybu pro naprostou většinu situací do kterých se může dostat.

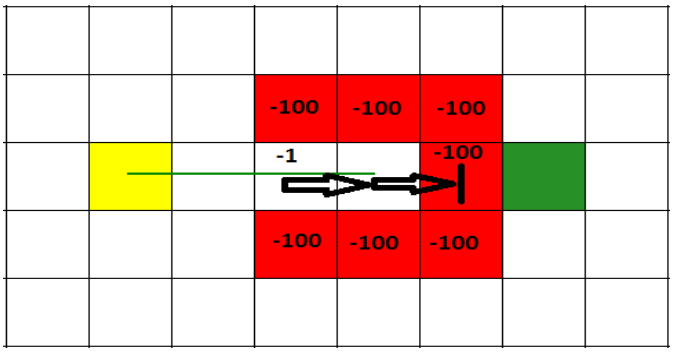
6.4.4 Nedostatky algoritmu

Algoritmus má bohužel problémy se zacyklením v lokálních minimech. To je způsobeno dělením stavů na zablokované a nezablokované. Pokud se algoritmus z nezablokovaného stavu dostane do lokálního minima (většinou slepé uličky tvořené konkávními překážkami), nejvyšší Q-hodnotu po několika iteracích bude mít akce vedoucí zpět do nezablokovaného stavu, což dostane robota do nekonečného cyklu.

Tento problém je možné řešit několika způsoby:

1. Ohodnocovat nezablokované stavy stejně jako zablokované, tzn. prohledávat všechny dostupné akce ve všech stavech. Toto řešení však mnohonásobně zvyšuje výpočetní čas pro naučení jednoho scénáře, což pro účely této práce nebylo vhodné
2. Přidat funkci, transformující konkávní překážky na konvexní
3. Nevystavovat robota prostředím s konkávními překážkami
4. Ukončení algoritmu při detekci zacyklení

Z časových důvodů bylo do práce bohužel implementováno pouze poslední navržené řešení. Algoritmus se tedy ukončí, pokud počet průchodů stavem překročí danou mezní hodnotu.



Obr. 43 Problém lokálního minima – zacyklení

Dalším nedostatkem algoritmu je neschopnost hledat cestu ve složitých prostředích, kdy je nutné se často a delší dobu pohybovat směrem dále od cíle. Příkladem takového prostředí jsou různá bludiště, spojené místnosti atd. Tato vlastnost je častá u lokálních plánovačů, především kvůli způsobu implementace odměňovací funkce, která hledá akci vedoucí co nejblíže k cíli, bez ohledu na to, zda tím směrem není například slepá ulička. Algoritmus totiž ve skutečnosti nehledá nejlepší cestu, ale postupuje směrem přímo k cíli a pouze se vyhýbá nalezeným překážkám. Algoritmus je tedy vhodný především do prostředí s menšími konvexními, ideálně pouze bodovými překážkami.

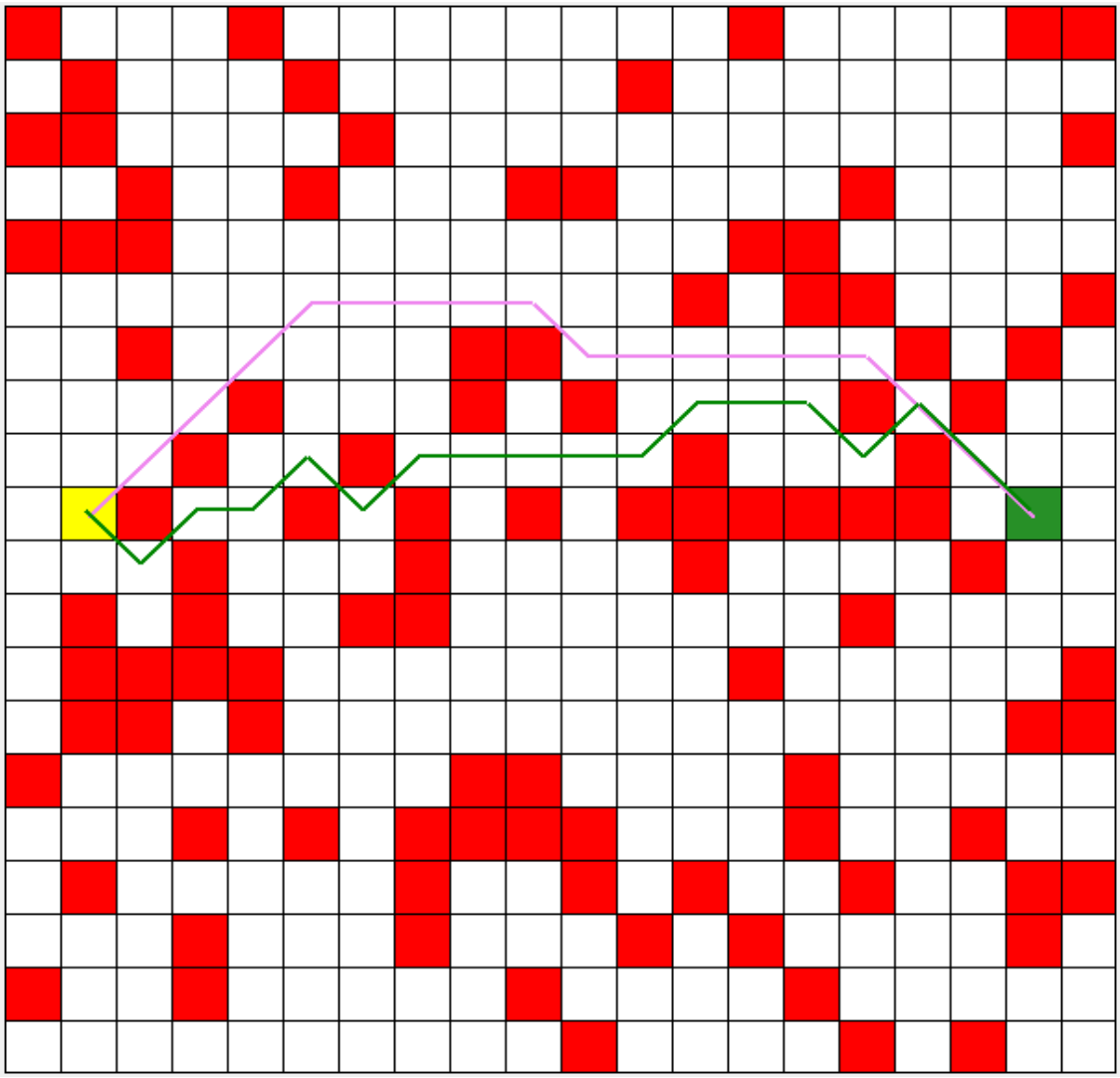
6.4.5 Generování cesty

Je třeba mít na paměti, že trénování strategie probíhá na naprosto náhodných scénářích. Existuje tedy určitá šance, že některé stavy ani po dokončení tréninku nejsou dostatečně prozkoumané, případně akce, která by jindy byla optimální v současném prostředí, optimálnost nezaručuje. Výsledná strategie tak často nemusí být optimální pro všechny typy prostředí, se kterými se robot může setkat. Proto je vhodné do algoritmu pro získávání cesty přidat schopnost „doučování“ pro konkrétní procházené prostředí.

Vzhledem k výše popsaném problému zacyklení je také nutné přidat schopnost částečných korekcí tohoto problému. Pokud by tedy zvolená akce vedla do stavu, jímž robot již jednou prošel, Q-hodnota akce se aktualizuje s odměnou -1 a následně se vybere nová akce. To se opakuje, dokud akce nevede do dosud nenavštíveného stavu, nebo nedojde k překročení maximálního počtu korekcí.

Popis algoritmu generování cesty:

1 Vlož startovní stav jako *s*
2 Opakuj, dokud *s* není cílový stav nebo není detekován cyklus
  2.1 Jestliže je *s* zablokovaný stav
    2.1.1 Najdi *s* v *Qpolicy*
    2.1.2 Pomocí greedy strategie vyber akci *a* a získej stav *s´*
    2.1.3 Jestliže již jednou prošel stavem *s´*, opakuj, dokud nenajde nový *s´:*
      2.1.3.1 Aktualizuj *Q(s,a)* dle vztahu (1) s *r = -1* a vyber novou akci *a*
    2.1.4 Aktualizuj *Q(s,a)* dle vztahu (1)
  2.2 Jestliže *s* není zablokovaný stav
    2.2.1 Vyber akci *a* vedoucí do stavu *s´* //směřující přímo k cíli



Obr. 44 Ukázka cesty naplánované pomocí lokálního plánovače (zeleně) a srovnání s algoritmem Q-učení pro statické prostředí (fialově)

Na příkladu je dobře vidět povaha takto generované trasy. Algoritmus se rozhoduje pouze na základě svého bezprostředního okolí. Volí tedy cestu vedoucí co nejpříměji k cíli, nicméně však naráží na spoustu překážek, které cestu ve výsledku prodlužují.

Naopak globální Q-učení pro statické prostředí dokáže najít skutečně optimální cestu, tak aby maximalizoval výslednou odměnu. Nevýhodou globálního plánování však samozřejmě je, že naučená strategie by pro jiné prostředí nefungovala.

6.4.6 Možnosti vylepšení algoritmu

Navržené řešení je stejně jako samotné Q-učení poměrně univerzální. S pouze malými uprávami by tedy mohlo být možné jej použít pro řešení dalších problémů, především pak pro plánování cesty v dynamickém prostředí.

Toho by pravděpodobně bylo možné dosáhnout rozlišováním tzv. „bezpečného“ stavu, kdy nemá ve svém okolí žádnou překážku, a „nebezpečného“ stavu, kdy má ve svém okolí alespoň jednu překážku. Úpravou odměňovací funkce tak, aby kladně odměňovala vstup do bezpečného stavu a záporně vstup do nebezpečného stavu, by mohla vzniknout strategie plánující cestu s dostatečným odstupem od okolních pohybujících se překážek. Tímto způsobem by se robot dokázal vyhnout většíně kolizí do doby, než by dorazil do cíle, za

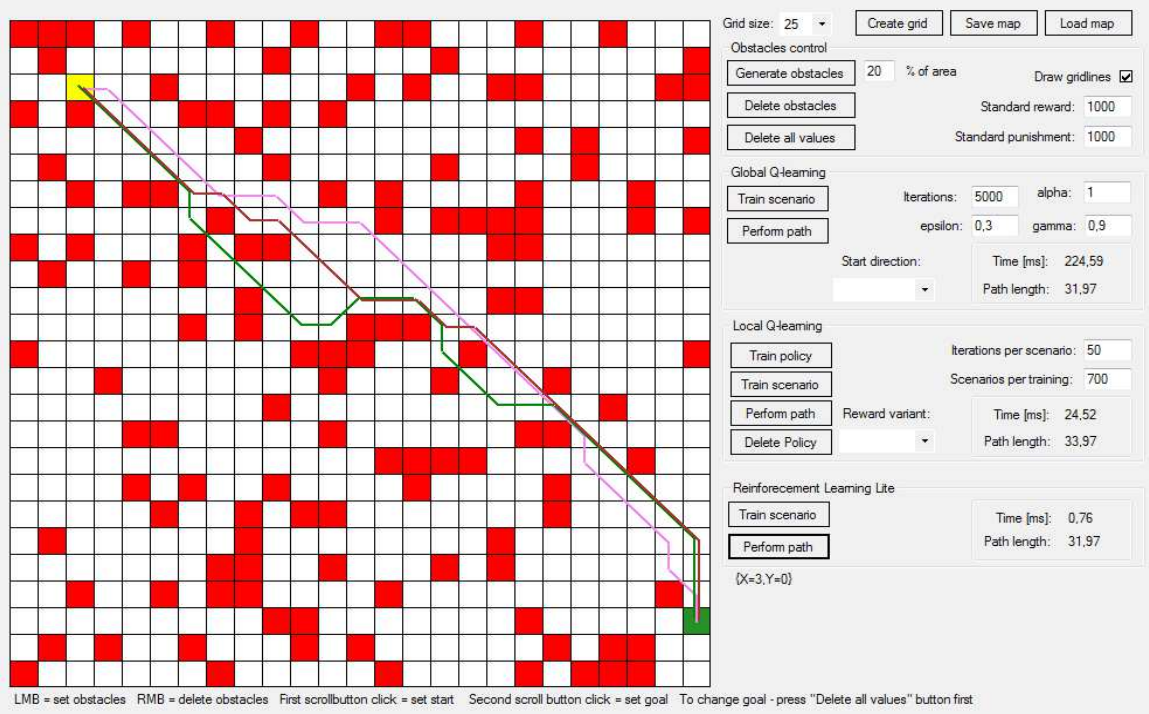
předpokladu, že rychlost překážek by byla menší nebo stejná jako rychlost robota.

Další možnost vylepšení by samozřejmě byla eliminace problému lokálního minima, ideálně transformací konkávních překážek na konvexní. Toho by šlo nejsnáze dosáhnout tak, že při detekci zacyklení by se překážky rozšířily i na problémové stavy.

6.5 Popis prostředí aplikace

Uživatelské prostředí bylo navrženo především s ohledem na okamžitou dostupnost všech ovládacích a informačních prvků. Aplikace je napsána v jazyce C# s využitím prvků .NET Framework 4.

6.5.1 Mapa a celkový pohled na interface



Obr. 45 Okno aplikace

Interface aplikace je tvořen dvěmi hlavními částmi. Na levé straně je čtvercová diskrétní mapa o velikosti 600x600 pixelů, pro kterou je možné nastavit různé množství a velikost jednotlivých buněk. Tato mapa slouží k zadávání překážek, startovací a cílové polohy a následně k vykreslení získaných trajektorií.

Barevný kód pro jednotlivé buňky mapy:

- Volné stavy – bílé
- Překážkové stavy - červené
- Startovací stav – žlutý
- Cílový stav – zelený

Barevný kód pro trajektorie generované jednotlivými algoritmy:

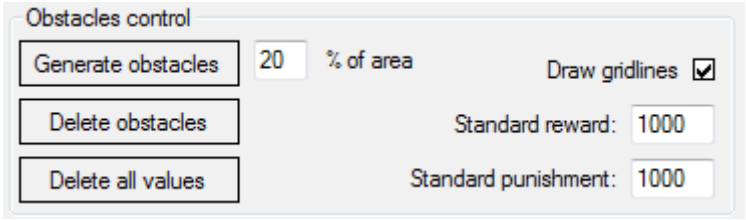
- Globální Q-učení pro statická prostředí - fialová
- Zjednodušené posilované učení - hnědočervená
- Lokální Q-učení pro neznámá prostředí - zelená

Na právé straně jsou pak všechny prvky potřebné pro nastavení mapy, ovládání algoritmů a zobrazování potřebných informací. V následujících podkapitolách jsou popsány jednotlivé části rozhraní aplikace.

6.5.2 Prvky pro vytváření map

Obr. 46 Tlačítka pro vytváření mřížky, ukládání a nahrávání uložených map

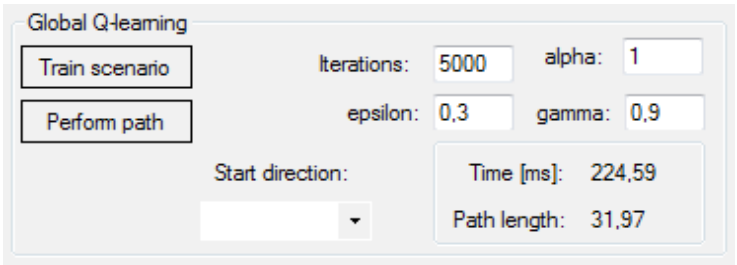
Pomocí tlačítek „Save map“ a „Load map“ je možné uložit aktuálně vytvořenou mapu do souboru pro pozdější opětovné využití. Do souboru se ukládá pouze poloha překážek, startu a cíle, žádné funkční hodnoty ani cesty uloženy nejsou.



Obr. 47 Prvky pro generování a mazání překážek, resetování hodnot, a nastavení odměn.

- *Generate obstacles* náhodně rozmístí překážky přibližně na zvolené procento stavů
- *Delete obstacles* smaže všechny překážky na mapě a resetuje všechny hodnoty stavů
- *Delete all values* resetuje všechny vygenerované cesty a fukční hodnoty použité pro jejich tvorbu
- *Standard reward a standard punishment* specifikují výši odměn a trestů za vstup na cílový/překážkový stav. Nemá vliv na Q-učení pro neznámá prostředí
- *Draw gridlines* umožňuje vypnout zobrazování mřížky na mapě

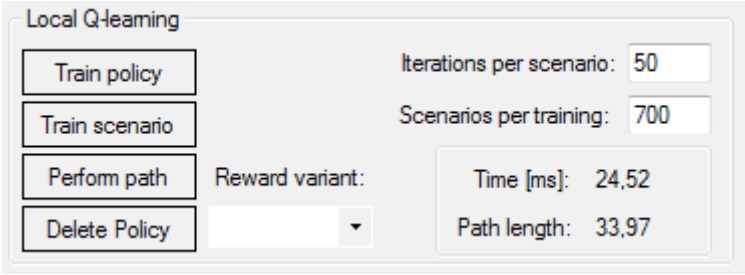
6.5.3 Ovládání globálního Q-učení pro statická prostředí



Obr. 48 Ovládací prvky pro Q-učení

- *Train scenario* spustí učení vytvořeného prostředí
- *Perform path* vykreslí cestu na základě Q-hodnot v naučeném prostředí
- *Iterations* nastavují, pro kolik epizod má být Q-učení spuštěno
- *Epsilon* specifikuje parametr ε v ε-greedy strategii
- *Alpha a gamma* nastavují parametry ze vztahu (1), pro všechny algoritmy
- *Start direction* nastavuje počáteční směr robota s omezením pohybu
- *Time* udává délku výpočtu strategie pro současné prostředí
- *Path length* udává délku získané trasy

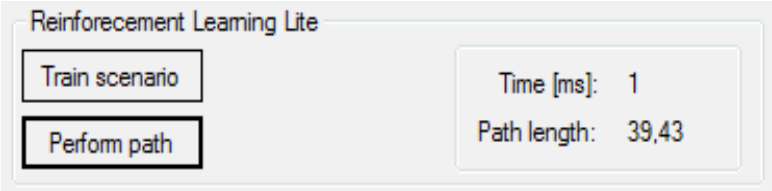
6.5.4 Ovládání lokálního Q-učení pro neznámá prostředí



Obr. 49 Ovládací prvky Q-učení pro neznámá prostředí

- *Train policy* spustí proces generování stanoveného počtu scénářů a učení strategie
- *Train scenario* spustí trénink strategie pouze pro současné prostředí
- *Perform path* vykreslí cestu k cíli na základě již vytvořené strategie
- *Delete policy* smaže všechny naučené strategie
- *Reward variant* určuje, která varianta odměňovací funkce bude použita, viz § 6.4.2
- *Iterations per scenario* určuje, kolikrát mají být jednotlivé scénáře naučeny
- *Scenarios per training* určuje, kolik různých scénářů má být vygenerováno při učení strategie
- *Time* udává délku výpočtu strategie pro současné prostředí nebo pro celou strategii
- *Path length* udává délku získané trasy
- Parametry alpha a gamma se nastavují jednotně pro všechny algoritmy, viz § 6.5.3

6.5.5 Ovládání zjednodušeného posilovaného učení pro statická prostředí

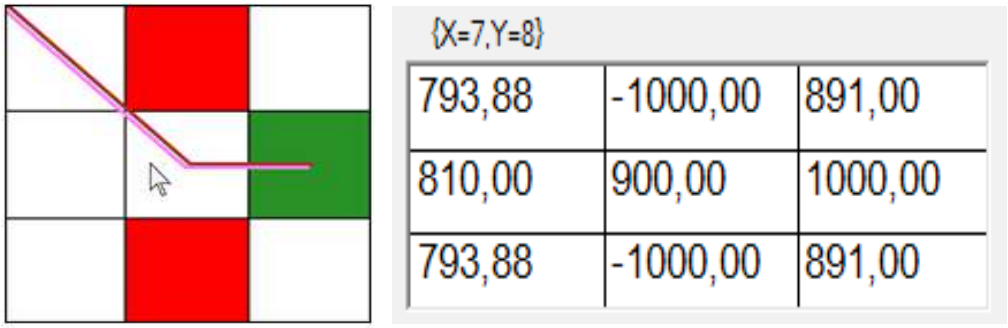


Obr. 50 Tlačítka pro ovládání zjednodušeného Q učení

- *Train scenario* spustí proces ohodnocování kvality stavů
- *Perform path* na základě kvality stavů vygeneruje nejkratší cestu k cíli

6.5.6 Informační panel hodnot stavů a akcí

Informační panel ukazuje pro algoritmus GQ Q-hodnoty a pro algoritmus RLL V-hodnotu stavu, nad nímž se nachází kurzor myši. Zobrazuje hodnoty pouze u globálních metod plánování cesty, ne však pro algoritmus lokálního Q-učení.



Obr. 51 Ukázka zobrazování Q-hodnot a V hodnoty pro zvolený stav.

Číslo uprostřed je V-hodnota určující kvalitu stavu pro zjednodušené posilované učení, okolní čísla jsou Q-hodnoty odpovídající jednotlivým akcí v naznačeném směru. Z ukázky je například vidět, že akce 4 „vpravo“ má nejvyšší Q-hodnotu, jelikož vede přímo k cíli.

6.5.7 Poznámky k ovládání

Zadávání překážek

Překážky lze vytvářet dvěma způsoby:

1. Automatickým vygenerováním překážek – zadáme kolik procent stavů by přibližně mělo být překážkových a stiskneme tlačítko *Generate obstacles.*
2. Manuálně pomocí kliknutí či tažením myši – při stisku a podržení levého tlačítka se jako překážky označí všechny stavy, přes které kurzor projede. Stejným způsobem je možné překážky i smazat při stistku pravého tlačítka.

Start a cíl

Start se zadává stisknutím scrollovacího kolečka myši. Druhé stisknutí kolečka nad jiným polem zadá cíl. Další stisknutí kolečka současný start i cíl odstraní a zadá nový start. Vyjímkou je situace, kdy je již vygenerována strategie či cesta kterýmkoliv z algoritmů – v tom případě je poloha cíle zafixována a nejde změnit ani odstranit, dokud nestiskneme tlačítko *Delete all values.*

Nastavení omezení pohybu u algoritmu GQ

Pokud je v ovládání algoritmu hodnota *Start direction* nenastavena a nebo nastavena na variantu „*9 none“*, algoritmus uvažuje všesměrového robota s až osmi dostupnými akcemi v každém stavu. Ve chvíli kdy nastavíme jakoukoliv jinou startovní orientaci, algoritmus začne uvažovat robota s maximálně třemi dostupnými akcemi. Při změně počáteční orientace se automaticky smažou všechny dosud aktualizované Q-hodnoty.

Volba odměňovací funkce u algoritmu LQ

Odměňovací funkci volíme pomocí seznamu hodnot *Reward variant.* Pokud je tato hodnota nenastavena, defaultně se použije první varianta odměňovací funkce.

Obě varianty lze za běhu programu libovolně měnit, aplikace si obě strategie ukládá, dokud program nevypneme nebo strategie sami nesmažeme.

6.5.8 Požadavky na PC

Pro správný běh programu je potřeba:

- PC s Windows XP či vyššími
- Myš s nepřemapovanou funkcí scrollovacího kolečka
- Monitor s minimálním rozlišením 1024x768

Rychlost algoritmů je ovlivněna výkonem PC, prezentované časy učení tedy nemusí být vždy stejné.

7 SROVNÁVACÍ EXPERIMENTY

7.1 Experiment 1

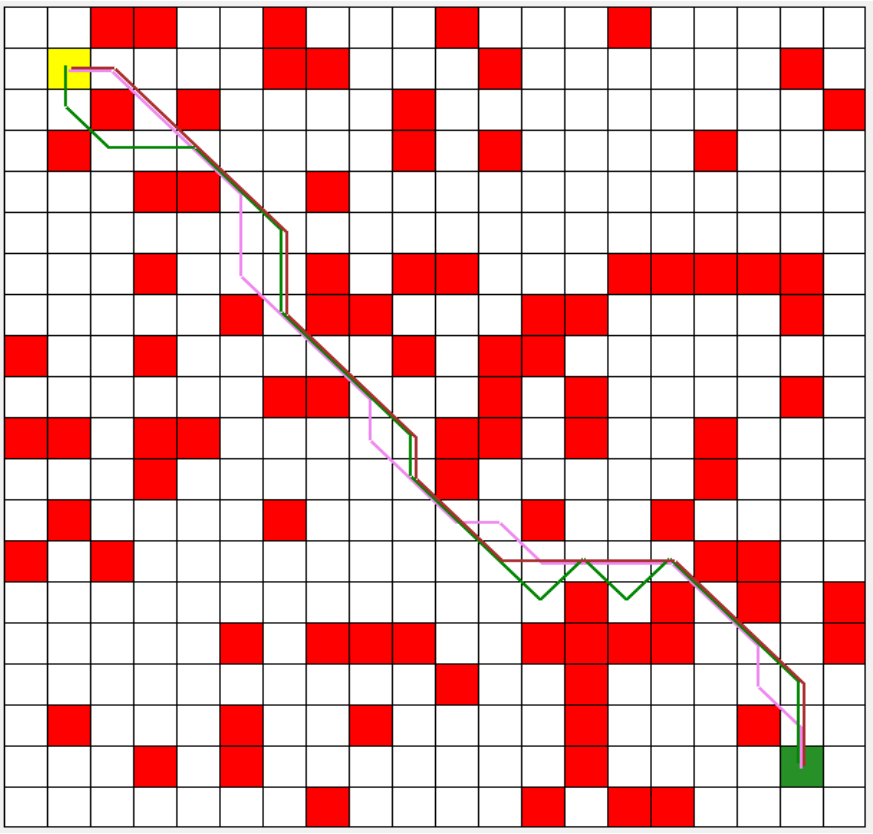
V tomto experimentu bylo provedeno srovnání výpočetních časů jednotlivých algoritmů a délky vygenerované trajektorie.

7.1.1 Popis experimentu

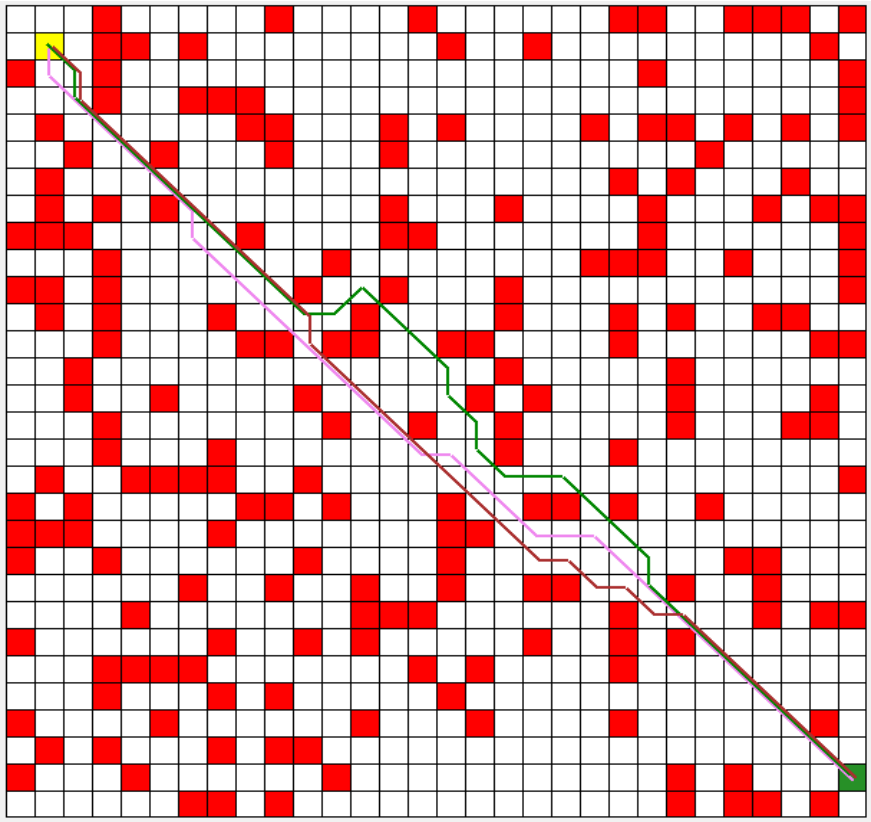
Pro testování algoritmů byly vytvořeny tři mapy o velikostech mřížky 20x20, 30x30 a 40x40. Následně byly náhodně vygenerovány překážky, start a cíl byly vloženy do diagonálně protějších rohů. V každém ze scénářů byly jednotlivé algoritmy osmkrát spuštěny. Při každém spuštění byly zapsány délka výpočtu pro daný scénář a délka vygenerované trajektorie. Výsledky byly následně zprůměrovány.

Pro všechny scénáře byly nastaveny jednotné parametry pro všechny algoritmy:
Globální Q-učení: $\alpha = 1$, $\gamma = 0{,}9$, $\varepsilon = 0{,}3$, počet iterací = 5 000
Lokální Q-učení: $\alpha = 1$, $\gamma = 0{,}9$, $\varepsilon = 0{,}3$, počet iterací na scénář = 50
Zjednodušené posilované učení: $\gamma = 0{,}9$

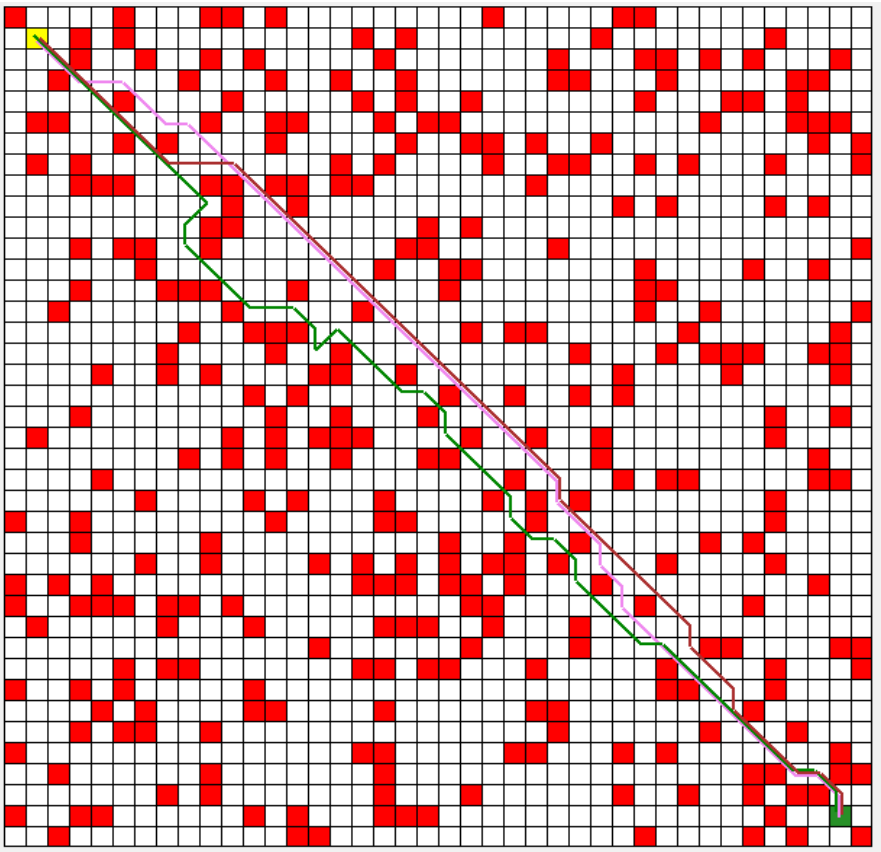
U lokálního Q-učení je třeba výrazně méně iterací, jelikož v nezablokovaných stavech není třeba nic ohodnocovat, a odměňovací funkce není závislá na dosažení cílového stavu.



Obr. 52 Ukázka scénáře 20x20. Fialová = GQ, Červená = RLL, Zelená = LQ



Obr. 53 Ukázka scénáře 30x30. Fialová = GQ, Červená = RLL, Zelená = LQ



Obr. 54 Ukázka scénáře 40x40. Fialová = GQ, Červená = RLL, Zelená = LQ

7.1.2 Výsledky experimentu

Legenda:

- GQ1 – globální Q-učení bez omezení pohybu robota
- GQ2 – globální Q-učení s omezením pohybu a počáteční orientací směrem k cíli
- RLL – zjednodušené posilované učení
- LQ1 – lokální Q-učení s první variantou odměňovací funkce
- LQ2 – lokální Q-učení s druhou variantou odměňovací funkce

Výsledné časy jsou udávány v milisekundách. Délky cesty je udána pouze jednou, jelikož se pro jednotlivé pokusy nemění.

Mapa:	20x20 - náhodně vygenerované překážky				
Algoritmus:	GQ1	GQ2	RLL	LQ1	LQ2
Iterace	5000	5000	1	50	50
Pokus č. 1	218,73	192,5	0,26	33,37	36,32
Pokus č. 2	214,67	190,35	0,43	25,39	18,86
Pokus č. 3	206,89	183,86	0,3	24,3	13,77
Pokus č. 4	220,69	180,82	0,42	25,49	16,93
Pokus č. 5	212,98	193,02	0,35	22,33	22,7
Pokus č. 6	219,29	198,38	0,4	34,5	12,5
Pokus č. 7	226,11	187,15	0,27	24,84	24,59
Pokus č. 8	221,6	200,18	0,32	23,56	35,26
Průměr:	**217,62**	**190,7825**	**0,34375**	**26,7225**	**22,61625**
Délka cesty:	**26,92**	**26,92**	**26,92**	**29,15**	**30,92**

Tab.1 Výsledné hodnoty pro první scénář

Mapa:	30x30 - náhodně vygenerované překážky				
Algoritmus:	GQ1	GQ2	RLL	LQ1	LQ2
Iterace	5000	5000	1	50	50
Pokus č. 1	337,09	250,37	0,92	16,8	12,68
Pokus č. 2	318,99	250,23	0,9	18,32	26,28
Pokus č. 3	334,62	251,46	0,62	22,51	12,95
Pokus č. 4	315,71	256,06	0,9	23,07	20,22
Pokus č. 5	314,96	244,64	0,94	26,25	27,53
Pokus č. 6	303,56	259,98	0,71	16,37	13,14
Pokus č. 7	313,17	239,15	0,92	17,6	12,36
Pokus č. 8	311,94	253,63	0,94	20,96	18,86
Průměr:	**318,755**	**250,69**	**0,85625**	**20,235**	**18,0025**
Délka cesty	**40,25**	**40,25**	**40,25**	**42,25**	**41,07**

Tab. 2 Výsledné hodnoty pro druhý scénář

Mapa:	40x40 - náhodně vygenerované překážky				
Algoritmus:	GQ1	GQ2	RLL	LQ1	LQ2
Iterace	5000	5000	1	50	50
Pokus č. 1	470,62	387,4	1,12	38,56	22,75
Pokus č. 2	465,7	382,57	1,79	25,49	39,91
Pokus č. 3	467,11	386,32	1,3	47,98	32,05
Pokus č. 4	454,92	375,28	1,66	39,91	41,38
Pokus č. 5	494,92	383,27	1,33	40,45	57,11
Pokus č. 6	489,25	367,22	0,79	35,89	33,53
Pokus č. 7	451,37	377,09	1,15	35,26	35,89
Pokus č. 8	464,71	377,64	1,64	46,86	33,48
Průměr:	**469,825**	**379,5988**	**1,3475**	**38,8**	**37,0125**
Délka cesty	**54,33**	**54,33**	**54,33**	**58,53**	**59,57999**

Tab. 3 Výsledné hodnoty pro druhý scénář

7.1.3 Zhodnocení výsledků

Z výše uvedených tabulek lze jasně vidět rozdíly mezi jednotlivými algoritmy:

- Globální Q-učení je ze všech algoritmů zdaleka nejpomalejší, zatímco zjednodušené posilované učení potřebuje k naučení prostředí pouhý zlomek času.
- Globální Q-učení s omezením pohybu je přibližně o 20% rychlejší než bez omezení pohybu, jelikož pro každý stav prozkoumává výrazně méně akcí.
- Lokální Q-učení v obou variantách je přibližně 12x rychlejší při učení jednoho scénáře než globální Q-učení – tato hodnota však není příliš důležitá, jelikož lokální plánovač si vytváří celkovou strategii pohybu předem a nemusí se tedy učit konkrétní scénář
- Globální Q-učení při dostatečném počtu iterací a vhodně nastaveném ε parametru zaručuje nalezení nejkratší cesty
- Zjednodušené Q-učení zaručuje nalezení nejkratší cesty vždy.
- Lokální Q-učení nalezení nejkratší cesty nezaručuje, odchylka od nejkratší cesty však obvykle není příliš velká.
- První varianta odměňovací funkce lokálního Q-učení potřebuje o 2-8% více času než druhá varianta při stejném počtu iterací, ale ve většině případů najde o něco kratší cestu.

Tyto výsledky podporují domněnku, že globální Q-učení je vhodné pouze pro případy, kdy plánujeme cestu s nějakým pohybovým nebo jiným omezením, případně máme speciální požadavky na trajektorii. Pro běžné situace je výrazně vhodnější použít jiný algoritmus, např. navržené zjednodušené posilované učení.

Lokální Q-učení je schopné natrénovat jeden scénář výrazně rychleji než globální Q-učení, nicméně po použití algoritmu pro kompletní naučení strategie již trénování daného scénáře není potřeba. Doba výpočtu je pak v takovém případě rovna nule, a algoritmus může rovnou generovat trajektorii. Hodnoty pro lokální algoritmus jsou tedy pouze informační.

7.2 Experiment 2

Experiment měl za cíl srovnání doby potřebné pro trénink kompletní strategie pohybu lokálního Q-učení při různých variantách odměňovací funkce a zhodnocení úspěšnosti a optimality generovaných trajektorií.

7.2.1 Popis experimentu

Nastavení parametrů lokálního Q-učení:

- $\alpha = 1$, $\gamma = 0{,}9$, $\varepsilon = 0{,}3$, počet iterací na scénář = 50, počet scénářů = 800

Bylo experimentálně ověřeno, že tyto parametry vedou k dostatačně prozkoumané strategii pohybu pro většinu dosažitelných stavů, při přijatelné době tréninku. Při vyšším počtu scénářů a iterací se kvalita hledaných cest a úspěšnost pravděpodobně zvýší.

Při měření času výpočtu bylo spuštěno učení obecné strategie pohybu s nastavenými parametry, vždy osmkrát pro každou variantu odměňovací funkce. Výsledky pak byly zprůměrovány pro přesnější srovnání.

Pro zhodnocení úspěšnosti při dosahování cíle a optimality trajektorií byla s výše uvedenými parametry vytvořena obecná strategie pohybu. Následně bylo vygenerováno třicet scénářů s velikostí 40x40 buněk a překážkami na přibližně 20% plochy a proběhlo nalezení trajektorie ve scénáři s oběmi variantami odměňovací funkce lokálního plánovače, a pro porovnání kvality cest i metodou zjednodušeného posilovaného učení (RLL). Umístění startu a cíle bylo stejné jako v experimentu 1. Zjišťovali jsme, zda byla nalezena cesta k cíli na první pokus, a jaká byla délka trajektorie.

7.2.2 Výsledky experimentu

Celková doba výpočtu strategie - výsledky jsou udávány v sekundách:

Algoritmus:	LQ1	LQ2
Pokus č. 1	18,79	17,16
Pokus č. 2	19,5	21,6
Pokus č. 3	27,91	16,89
Pokus č. 4	21,59	24,06
Pokus č. 5	28,532	16,04
Pokus č. 6	23,46	25,57
Pokus č. 7	19,79	17,19
Pokus č. 8	22,9	18,767
Průměr:	**22,809**	**19,65963**

Tab. 4 Výsledné časy učení kompletní strategie pohybu lokálním Q-učením

Jak je vidět, výsledné časy korespondují s výsledky z experimentu 1. První varianta odměňovací funkce vyžaduje průměrně o 16% delší čas tréninku pro stejné počty iterací.

Mapa:	40x40 - náhodně generované překážky						
Algoritmus:	LQ1	LQ2	RLL	Algoritmus	LQ1	LQ2	RLL
Iterace:	50x800	50x800	1	Iterace:	50x800	50x800	1
Pokus č. 1	55,12	63,12	54,53	Pokus č.16	54,53	59,35	54,53
Pokus č. 2	55,94	62,76	54,53	Pokus č.17	61,35	-	53,94
Pokus č. 3	55,71	56,53	53,94	Pokus č.18	-	-	53,35
Pokus č. 4	56,89	61,94	55,12	Pokus č.19	58,53	58,53	55,35
Pokus č. 5	68,76	60,76	55,15	Pokus č.20	55,35	59,71	53,35
Pokus č. 6	58,53	57,12	54,53	Pokus č.21	56,17	55,35	54,53
Pokus č. 7	58,89	68,89	53,94	Pokus č.22	54,76	56,76	53,94
Pokus č. 8	56,53	60,3	55,12	Pokus č.23	53,94	54,53	53,94
Pokus č. 9	53,35	53,35	52,76	Pokus č.24	56,53	55,35	53,94
Pokus č. 10	56,76	-	53,94	Pokus č.25	55,71	-	54,53
Pokus č. 11	56,53	56,53	53,94	Pokus č.26	53,35	55,35	53,35
Pokus č. 12	57,12	59,94	55,12	Pokus č.27	56,53	55,94	53,94
Pokus č. 13	53,35	53,35	53,35	Pokus č.28	55,12	59,12	54,53
Pokus č. 14	57,71	62,3	54,53	Pokus č.29	54,53	56,53	53,94
Pokus č. 15	55,35	65,71	53,94	Pokus č.30	55,94	-	54,53

Tab. 5 Délky trajektorií pro jednotlivé náhodně generované scénáře

7.2.3 Zhodnocení výsledků

	LQ1	LQ2	RLL
Průměrná délka trasy	56,51	58,76	54,20
Odchylka délky trasy od optima	4,20%	8,41%	0%
Procentuální úspěšnost	96,70%	83,30%	100%

Tab. 6 Statistické zhodnocení výsledků

Z uvedených výsledků je patrné, že první varianta odměňovací funkce má o několik procent delší dobu výpočtu strategie. Generuje však v průměru o něco kratší trajektorie a má výrazně vyšší úspěšnost při dosahování cíle na první pokus.

Je nutné zmínit, že díky doučovacímu algoritmu při generování cesty by se úspěšnost a kvalita cest zvýšila při několikanásobném procházení stejného scénáře. Obvykle jediné případy, kdy robot nedojde do cíle, jsou případy lokálních minim u větších konkávních překážek, a nebo nevhodných typů prostředí (bludiště atd.).

7.3 Experiment 3

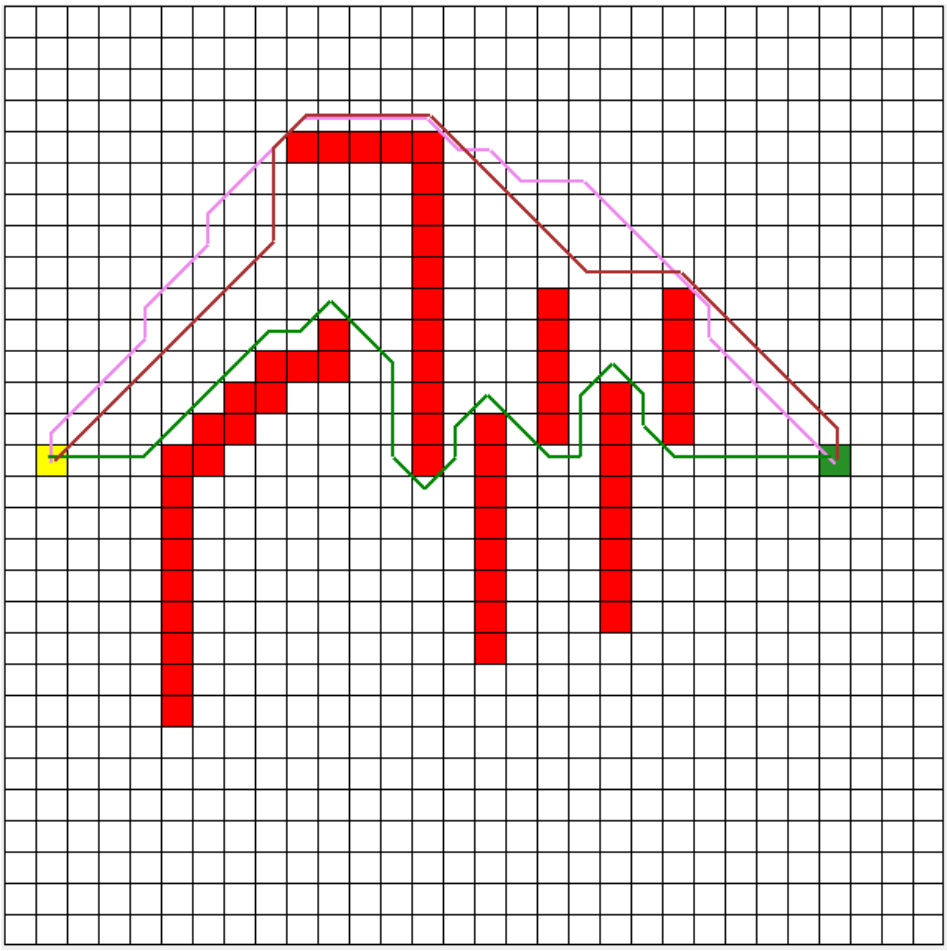
Tento experiment zahrnuje doplňující experimenty pro ověření schopnosti algoritmů plánovat cesty v různých typech prostředích. Scénáře měly za cíl simulovat vnitřní i vnější prostředí s různými typy překážek. Všechny algoritmy byly nastaveny stejným způsobem jako v experimentech 1 a 2. Lokální plánovač používal první variantu odměňovací funkce.

7.3.1 Popis a výsledky experimentu

Pro tento experiment byla použita ručně vytvořená prostředí, napodobující určité typické situace.

1. **Prostředí typu „překážková dráha“**

Toto prostředí simulovalo určitý typ překážkové dráhy. Účelem tohoto prostředí bylo ověřit schopnost lokálního Q-učení najít cestu přes složitý terén, a demonstrovat rozdílnost tras generovaných lokálním a globálním plánovačem.



Obr. 55 Trajektorie generované algoritmy pro prostředí typu překážková dráha Fialová = GQ, Červená = RLL, Zelená = LQ

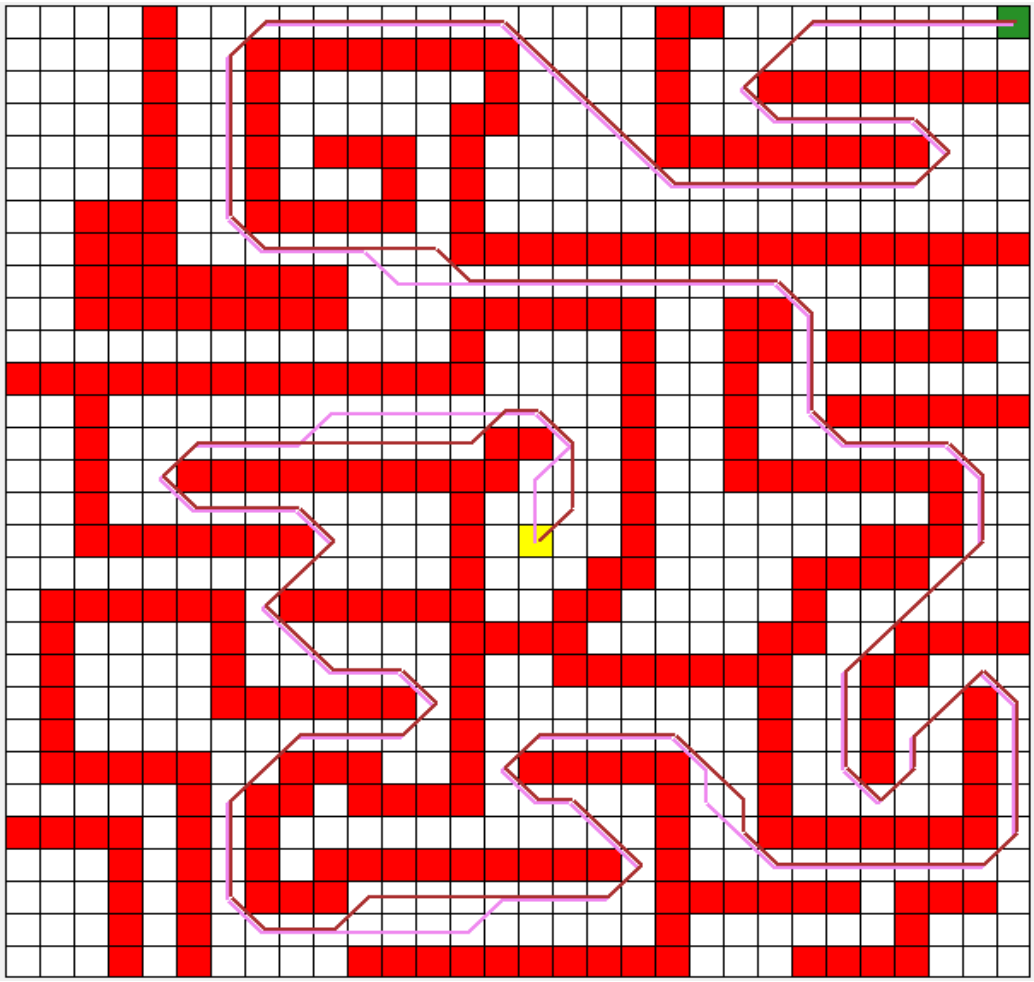
Algoritmus	GQ1	RLL	LQ1
Průměrný výpočetní čas	559,068	0,998	0
Délka cesty	36,38	36,38	38,15

Tab. 7 Výsledné hodnoty hledání cesty v prostředí typu překážková dráha

Pro dané prostředí byla potvrzena schopnost všech algoritmů najít cestu k cíli i v komplikovaném prostředí. Opět se ukazuje velice vysoká výpočetní náročnost globálního Q-učení. Trajektorie generovaná lokálním plánovačem se od ostatních dle očekávání velice odlišuje. Agent se snaží jít přímo směrem k cíli a pouze se vyhýbá nalezeným překážkám na základě své obecné strategie pohybu.

2. Prostředí typu „bludiště“

Toto prostředí bylo vytvořeno pro demonstraci schopnosti globálních plánovačů nalézt cestu i ve velice komplikovaném a složitém prostředí.



Obr. 56 Trajektorie generované globálními plánovači v prostředí typu bludiště. Fialová = GQ, Červená = RLL

Algoritmus	GQ1	RLL
Průměrný výpočetní čas	1347,4	0,786
Délka cesty	172,5	172,5

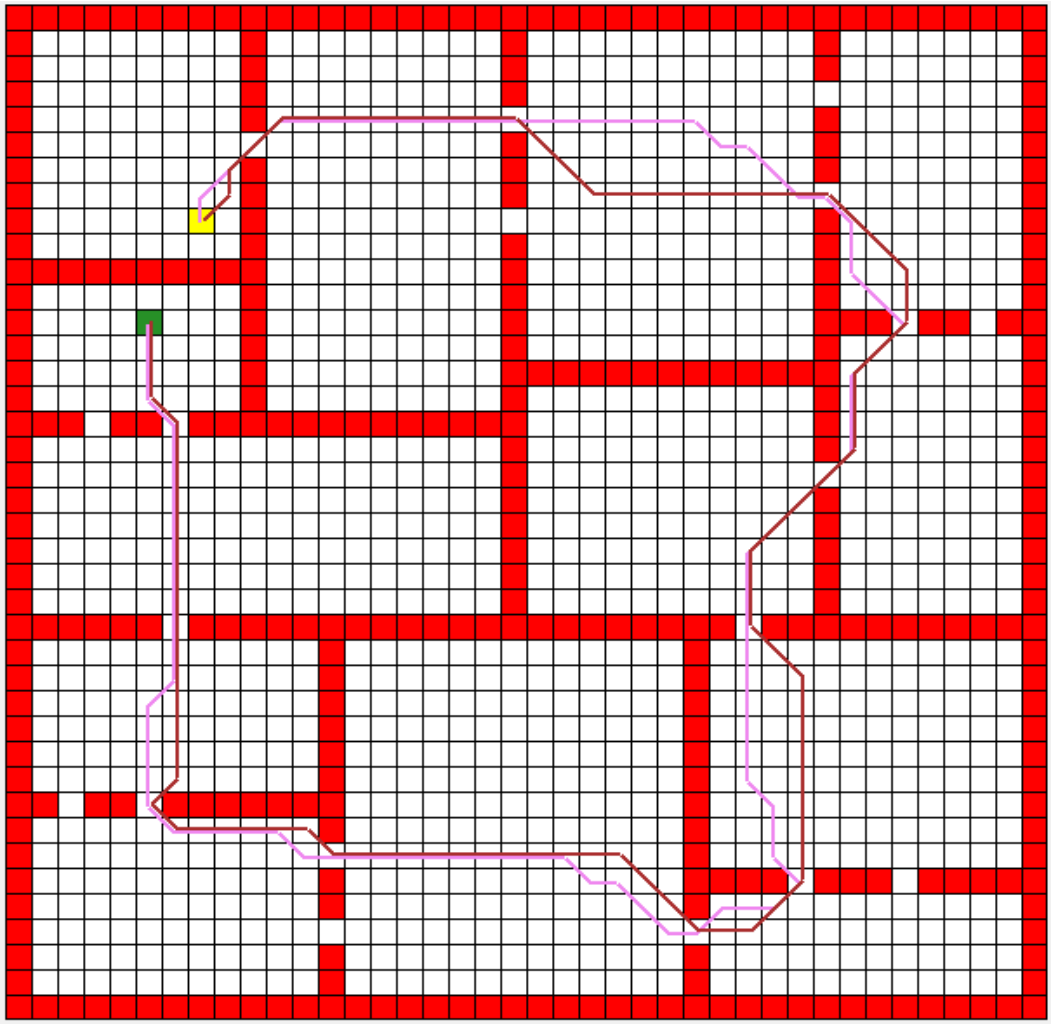
Tab. 8 Výsledné hodnoty pro prostředí typu bludiště

Za povšimnutí stojí výrazně delší doba výpočtu globálního Q-učení, proti době výpočtu stejného algoritmu na mapě 30x30 z experimentu 1. Přestože je celkový počet stavů stejný, výpočetní doba se zvyšuje i v závislosti na celkové délce trajektorie. Naproti tomu zjednodušené Q-učení má téměř konstantní časovou náročnost pro daný počet stavů, v tomto případě je tak dokonce více než 1500x rychlejší.

V takovémto typu prostředí implementované lokální Q-učení naprosto selhává, proto není testováno.

3. Prostředí – místnosti

Toto prostředí bylo převzato z práce [8] pro ověření schopnosti hledání cesty v prostředí několika spojených místností s více průchody.



Obr. 57 Trajektorie generované globálními plánovači v prostředí spojených místností Fialová = GQ, Červená = RLL

Algoritmus	GQ1	RLL
Průměrný výpočetní čas	960	1,25
Délka cesty	106,66	106,66

Tab. 8 Výsledné hodnoty

7.3.2 Zhodnocení experimentu

Experiment potvrdil schopnost globálních plánovačů najít cestu ve všech typových prostředích, s různými druhy překážek a uspořádání. Lze také nalézt spojitost mezi dobou výpočtu Q-učení a celkovou délkou trajektorie.

Experiment potvrdil také omezení lokálního plánovače, který je schopen hledat cestu pouze v prostředích s konvexními překážkami. Na složitějších prostředích typu bludiště či místností, kdy optimální cesta vede jinou cestou než přibližně přímo k cíli, algoritmus téměř vždy selhává.

8 ZÁVĚR

Mým hlavním úkolem bylo navrhnout a implementovat systém pro plánování cesty robota pomocí posilovaného učení. Po důkladné rešerši jsem se rozhodl implementovat tři velice odlišné algoritmy demonstrující široké možnosti využití posilovaného učení při plánování pohybu robota.

Prvním z nich je algoritmus pro globální plánování cesty ve statickém prostředí pomocí klasického Q-učení. Tento algoritmus naplňuje hlavní předpoklady ohledně Q-učení – při dostatečném počtu iterací a správně nastaveném prohledávání nalezne optimální trajektorii v libovolném typu prostředí. Bohužel jeho výpočetní náročnost je poměrně vysoká, především pak pro rozsáhlé stavové prostory. Hlavní využití tohoto algoritmu je tedy především v situacích, kdy máme na naplánované trajektorie nebo způsob pohybu robota nějaké speciální požadavky, např. neholonomní omezení pohybu, nebo požadavek minimálního počtu změn směru. Pro klasický případ plánování cesty všesměrového robota ve známém prostředí je však pravděpodobně vhodnější použít tradiční přístupy k plánování cesty z důvodu výrazně vyšší časové efektivity.

Druhé navržené řešení je globální algoritmus zjednodušeného posilovaného učení pro statická prostředí. U tohoto algoritmu jsem původně vycházel z Q-učení, je však upraven tak, aby nebylo třeba velkého množství iterací jako u klasického Q-učení. Díky tomu stačí hodnoty pro všechny stavy aktualizovat pouze jednou při vhodně zvoleném způsobu učení prostředí. Mezi hlavní výhody patří mnohonásobně nižší výpočetní náročnost, schopnost nalézt cestu do cíle z libovolného místa na mapě a vždy zaručená optimalita nalezené trajektorie.

Třetím algoritmem je systém pro lokální plánování cesty založený na Q-učení. Tento algoritmus nemá za úkol najít vždy nejkratší možnou cestu do cíle, ale spíše se vyhýbat nalezeným překážkám v neznámém prostředí při své cestě k cíli. Tento algoritmus používá Q-učením natrénovanou obecnou strategii pohybu, již pak aplikuje na neznámé prostředí při pohybu směrem k cíli. V aplikaci jsou implementovány dvě varianty odměňovací funkce pro Q-učení, přičemž každá má poněkud jiné vlastnosti a plánuje jiné trajektorie, celkově však dosahují dobrých výsledků, především pak při dodržení podmínek pro tvar a velikost překážek.

Všechny algoritmy jsou navrženy pro fungování v jednotné diskrétní mapě, což vedlo k několika kompromisům a občas nižší efektivitě, v zásadě by však navržená řešení měla být s mírnými úpravami použitelná i pro specifické praktické situace.

V práci je popsáno několik experimentů, které dostatečně demonstrují výhody a nevýhody jednotlivých navržených řešení. Na výsledcích experimentů je vidět, že každý algoritmus je vhodný pro jiné situace, generuje jiné trajektorie a mají rozdílné způsoby využití v odlišných typech prostředí. Jako celek však navržená řešení dokazují hlavní výhody posilovaného učení, a to jeho univerzálnost, přizpůsobitelnost, snadnou implementaci a širokou variabilitu využití v různých situacích.

Jako možnosti rozšíření práce bych navrhoval především úpravu navrženého algoritmu lokálního plánování tak, aby byl schopen navigovat robota v dynamickém prostředí s pohyblivými překážkami. Další možnosti rozšíření by bylo přizpůsobení navržených algoritmů pro konkrétní typy robotů a prostředí, jelikož navržená řešení jsou koncipována spíše obecně a nezahrnují různá omezení vyplývající z hardwarové výbavy, dosahu sensorů a možností pohybu robota.

SEZNAM POUŽITÉ LITERATURY

[1] JARADAT,M. A. K.; AL-ROUSAN, M.; QUADAN, L. Reinforcement based mobile robot navigation in dynamic environment. Robotics and Computer-Integrated Manufacturing, vol. 27, 2011, pp. 135–149.

[2] BARRIOS-ARANIBAR, D; ALSINA, P. J. Reinforcement Learning-Based Path Planning for Autonomous Robots. In I ENRI - Encontro Nacional de Robótica Inteligente no XXIV Congresso da Sociedade Brasileira de Computação, Salvador, BA, Brazil, 2004.

[3] ŠVEC, P. *Plánování trasy robota pomocí voroného diagramů a dalších prostředů výpočetní geometrie* [pdf]. Brno, 2006 [cit. 2013-02-20]. Dostupné z: http://is.muni.cz/th/39233/fi_m/diplomka_Petr_Svec.pdf. Diplomová práce. MASARYKOVA UNIVERZITA V BRNĚ FAKULTA INFORMATIKY.

[4] ŠINDELÁŘ, J. *Plánování cesty neholonomního mobilního robotu* [pdf]. Brno, 2010 [cit. 2013-02-20]. Dostupné z: http://autnt.fme.vutbr.cz/szz/2010/DP_Sindelar.pdf. Diplomová práce. VUT Brno.

[5] ŠÍMA, M. *Plánování cesty robota ve spojitém prostředí* [pdf]. Brno, 2006 [cit. 2013-02-20]. Dostupné z: http://autnt.fme.vutbr.cz/szz/2006/DP_Sima.pdf. Diplomová práce. VUT Brno.

[6] HLAVÁČ, V. *Motion planning methods* [pdf]. Prague, 2010 [cit. 2013-02-20]. Dostupné z: http://cmp.felk.cvut.cz/~hlavac/TeachPresEn/55IntelligentRobotics/100MotionPlanningMethods.pdf

[7] EDEN, T.; KNITTEL, A.;VAN UFFELEN, R. *Reinforcement learning* [online]. [cit. 2013-02-27]. Dostupné z: http://www.cse.unsw.edu.au/~cs9417ml/RL1/

[8] VESELÝ, P. Navigace robota pomocí případového usuzování a posilovaného učení [pdf]. Brno, 2005 [cit. 2013-02-27]. Dostupné z: http://uloz.to/xXFF5ZA/veselydp-pdf. Diplomová práce. UAI VUT Brno.

[9] SUTTON, R. S.; BARTO, A.G. *Reinforcement Learning: An Introduction* [online]. The MIT Press, 2002, 2005 [cit. 2013-02-28]. Dostupné z: http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html

[10] BŘEZINA, T.; DVOŘÁK, J. Algoritmy / prostředky umělé inteligence: Automatický klasifikátor/aproximátor. [online]. [cit. 2013-02-28]. Dostupné z: http://uloz.to/xX6btq1/vai-9-pdf

[11] DLOUHÝ, M. Pravděpodobnostní plánování. *Robotika.cz* [online]. 5.12.2003 [cit. 2013-05-07]. Dostupné z: http://robotika.cz/guide/probplan/cs

[12] STÁREK, I. Plánování cesty robota pomocí dynamického programování [pdf]. Brno, 2009 [cit. 2013-05-07]. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/10107/Starek_DP_2009.pdf?sequence=1

[13] GE S.;CUI Y. Dynamic motion planning for mobile robots using potential field method. Autonomous Robots, 2002;[cit. 2013-05-07]

[14] BISHOP, A. *Allan Bishop Blog: Q-learning* [online]. 2010 [cit. 2013-05-13]. Dostupné z: http://blog.allanbishop.com/q-learning/

[15] CHEN, X. *Reinforcement Learning Overview: Grid World* [online]. 2006 [cit. 2013-05-13]. Dostupné z: http://www2.hawaii.edu/~chenx/ics699rl/grid/gridworld.html

[16] POOLE, D. *Q-learning demonstration* [online]. 2004 [cit. 2013-05-13]. Dostupné z: http://www.cs.ubc.ca/~poole/demos/rl/q10.html

[17] MASSON, T. *Artificial Intelligence - The Q-Learning Algorithm* [online]. 2003 [cit. 2013-05-13]. Dostupné z: http://thierry.masson.free.fr/IA/en/qlearning_about.htm

[18] DAS, P. K.; MANDATA,S.C.; NEBERA H.S; PATRO, S.N. *An Improved Q-learning Algorithm for Path-Planning of a Mobile Robot* [online]. 2012 [cit. 2013-05-15]. Dostupné z: http://research.ijcaonline.org/volume51/number9/pxc3881468.pdf

[19] KELLEHER, J. D. Robot Path Planning. [online]. [cit. 2013-05-15]. Dostupné z: http://www.comp.dit.ie/jkelleher/rtf/classmaterial/week9/RobotPathPlanning.pdf

[20] GROSS, T. Plánování cesty mobilního robota. Brno, 2007. Dostupné z: http://autnt.fme.vutbr.cz/szz/2007/DP_Gross.pdf. Diplomová práce.

[21] Geometric Reasoning and Applications: Robot Motion Planning. *USC Student Computing Facility* [online]. 1999 [cit. 2013-05-22]. Dostupné z: http://www-scf.usc.edu/~peiyingc/gra_planning.html

[22] Algorithms for Planning and Control of Robot Motion: Time-Optimal Trajectories for Differential Drive Robots. *USC Student Computing Facility*[online]. [cit. 2013-05-22]. Dostupné z: http://www.cse.unr.edu/robotics/tc-apc/dd_trajectories

[23] AutoParking-Small Vehicle. *Society of robots* [online]. 1999 [cit. 2013-05-22]. Dostupné z: http://www.societyofrobots.com/member_tutorials/book/export/html/344

[24]Zhang, T.; Zhu, Y.; Song, J. Real-time motion planning for mobile robots by means of artificial potential field method in unknown environment, Industrial Robot: An International Journal, Vol. 37 Iss: 4, pp.384 - 400
[25]Motion Planning in Robotics. [online]. [cit. 2013-05-22]. Dostupné z: http://www-cs-faculty.stanford.edu/~eroberts/courses/soco/projects/1998-99/robotics/basicmotion.html
[26]*Geometric computing lab* [online]. [cit. 2013-05-22]. Dostupné z: http://gclab.kaist.ac.kr/research.html
[27]Path Finding using Rapidly-Exploring Random Tree. *Nakkaya dot com* [online]. [cit. 2013-05-22]. Dostupné z: http://nakkaya.com/2011/10/27/path-finding-using-rapidly-exploring-random-tree/